jamk.fi

Mobiilisovelluskehitys Vue Nativella

Miikka Mäkelä

Opinnäytetyö
Toukokuu 2020
Tekniikan ala
Insinööri (AMK), tieto- ja viestintätekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Mäkelä, Miikka	Opinnäytetyö, AMK	Toukokuu 2020
	Sivumäärä 79	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Mobiilisovelluskehitys Vue Nativella

Tutkinto-ohjelma
Insinööri (AMK), tieto- ja viestintätekniikka

Työn ohjaaja(t)
Ari Rantala

Toimeksiantaja(t)
Zaibatsu Interactive Oy

Tiivistelmä

Opinnäytetyön tavoitteena oli tutustua Vue Nativeen ja luoda uusi versio vanhentuneesta mobiilisovelluksesta. Vue Native on mobiilikehityksessä hyödynnettävä kirjasto, jonka koodi kääntyy lopulta React Native koodiksi. Tämä säästää aikaa ja yhdellä koodipohjalla voidaan luoda iOS ja Android -sovelluksia. Pienet eroavaisuudet tulee kuitenkin ottaa huomioon. Zaibatsu Interactive Oy:llä oli kiinnostusta kuulla ja kokea, onko Vue Native -teknologiasta tulevaisuudessa hyväksi valinnaksi projekteihin. Sovelluksen tuli olla helppokäyttöinen ja selkeä. Siihen tuli myös uudistaa ulkoasu vastaamaan uudistettua brändi-imagoa.

Suunnittelun aikana käytettävyys ja sijoittelu nousivat tärkeiksi osa-alueiksi. Suunnitelma toteutettiin hyödyntäen Affinity-ohjelmia ja ottamalla vaikutteita menestyneistä mobiilisovelluksista. Vaatimukset sovellukselle kirjattiin Trelloon, jota käytettiin prosessinvalvomiseen ja -hallintaan. Hyvän suunnittelun ansiosta pystyttiin karsimaan virheellisiä ajatuksia heti alusta lähtien.

Toimeksiantajan tietotaito nousi arvoonsa, kun piti keksiä erilaisia keinoja ongelmiin. Kehityksen aikana luotiin myös testitietokanta, johon päädyttiin hyödyntämään NoSQL-tietokantaa nimeltä MongoDB. Datan säilömisen rakenne oli yksinkertainen, joten siitä ei seurannut erityisiä ongelmia. Lopputuloksena syntyi toimiva kokonaisuus, joka näyttää hyvältä ja tuntuu nykyaikaiselta. Vue Nativen soveltuminen mobiilisovelluskehityksessä todettiin toimivaksi. Vue Native -dokumentaatio on kuitenkin rajallinen ja vaatii React Native osaamista tueksi. Opinnäytetyön tuloksena oli toimiva ja näyttävä sovellus. Vue Nativen toimivuudesta ja sovellusrakentamisen selkeydestä saatiin käytännön kokemusta. Sovellusta ei rajoitteitten takia saatu kuitenkaan liitettyä yrityksen toimintaan lopullisesti.

Avainsanat (asiasanat)
Vue Native, Native, mobiilisovelluskehitys, React, Vue, Node, JavaScript, mobiilisovellus

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Mäkelä, Miikka	Bachelor's thesis	May 2020
		Language of publication: Finnish
	Number of pages: 79	Permission for web publication: x

Title of publication

Mobile Application Development with Vue Native

Degree programme

Information and Communication Technology, Bachelor's Degree in Engineering

Supervisor(s)

Rantala, Ari

Assigned by

Zaibatsu Interactive Oy

Abstract

The aim of the thesis was to find out how Vue Native can be utilized in mobile application development and remake an older version of a mobile application using Vue Native. Vue Native is a mobile development framework that uses React Native's API to output the codebase. In other words, Vue Native compiles into React Native. The benefit of this technology is, that you can develop your application for iOS and Android with minimal necessary code changes. Vue Native allows the developer to make use of the simpler ideas that Vue consists of, while getting many of the functionalities already provided by React Native.

Zaibatsu Interactive Oy needed to map the future of Vue Native against its close relative React Native. To demonstrate the use, an older application was to be remade using Vue Native. The requirements for the application were gathered onto Trello, which is used for project management. It was a useful step in order to keep a clear view of the full process. The design was carried out with Affinity programs and it was influenced by successful applications on the market.

In the development process, the user's needs were kept in mind. Not only should the application look great, but it should also work as intended and without confusion. Difficulties stemmed from the developer's inexperience with the use of Vue Native.

The application was completed with the wanted functionality and a modern look and feel. Vue Native was found to be a working piece of technology, which does require the user to know quite a few things from Vue and React. The documentation for Vue Native is limited and does not go into the much-needed depth. To get the most benefit, developers need to dive into the React Native documentation and learn some of the underlying patterns. This can make it more difficult than intended.

Keywords/tags (subjects)

Mobile development, Vue Native, Vue, React, Node, JavaScript

Miscellaneous (Confidential information)

Sisältö

Kuviot

1 Johdanto

1.1 Toimeksiantaja

Opinnäytetyössä toimeksiantajana toimi Zaibatsu Interactive Oy. Zaibatsu Interactive Oy on jyväskyläläinen vuonna 2014 perustettu yritys, joka tarjoaa frontend-ohjelmointia, alihankintaa ja edistää mobiilipelikehitystä. Yrityksellä on noin 20 työntekijää ja kasvu on ollut hyvää viime vuosina. Yrityksen tilat sijaitsevat Jyväskylän ydinkeskustassa. Asiakkaita ja yhteystyökumppaneita Zaibatsulta löytyy ympäri Suomea ja ulkomaillekin on rakennettu toimivia suhteita.

1.2 Lähtökohdat

Zaibatsulla on aikaisemmin ollut käytössä omien apk ja ipa -tiedostojen rakentamiseen hyödynnetty mobiilisovellus, joka ei ole ollut käytössä vähään aikaan. Toimeksiantajan suurin toive oli saada tietoa Vue Native -kirjastosta, jonka avulla voidaan luoda täysin native-ominaisuuksia hyödyntäviä mobiilisovelluksia. Näihin hyötyihin kuuluvat esimerkiksi korkea ruutupäivitysnopeus, joka varmistaa, että sovellus toimii sulavasti. Omat native-versiot laitteille saavat kaiken hyödyn irti puhelimen toiminnasta, joka kasvattaa suorituskykyä entisestään (Gillis 2018). Frontend-puoli sovellukselle rakennettiin uudestaan alusta asti, ottaen osittaisia vaikutteita hyväksi todetuista asetteluista ja toiminnoista. Yrityksen uusittu brändi-imago haluttiin myös näkyviin uudessa sovelluksessa.

Koska Vue Native perustuu suurimmaksi osaksi React Native API:iin, oli tarvittavaa tutustua molempii ja verrata samalla Vue ja React -kirjastoja. Myös MongoDB:n toimivuutta ja sen helppoa muutoksiin taipumista haluttiin hyödyntää. Zaibatsu Interactivella on ollut käytössä MySQL-tietokanta, jonka päivittäminen on koettu raskaaksi.

1.3 Kehitystyö

Työn tarkoituksena oli kehittää frontend mobiilisovellukselle ja demonstroida sen toimintaa luomalla vaadittava backend, joka tulee tulevaisuudessa korvaamaan nykyisen ratkaisun. Työssä suunniteltiin ja toteutettiin mobiilisovellus Android-käyttöön Vue Nativella. Työ oli tutkimuksellista kehitystyötä. Tutkimuksellinen kehitystyö yhdistää asioiden tutkimisen ja niiden soveltamisen käytännössä. Tällaisen työn päämotivaationa on parantaa jonkin asian laatua ja tehostaa sen toimintaa. (Työelämän tutkiva kehittämistoiminta n.d.)

2 Käytetyt teknologiat

2.1 Frontend

Frontend sisällyttää kaiken mitä palvelun tai tuotteen pinnalla liikkuu. Napit, animaatiot, tekstit ja asettelu ovat tässä tärkeimpiä osia. Frontendin hahmottaminen voi olla kuitenkin hankalaa, ja useat yritykset omaavat täysin omanlaisensa käsityksen siitä, mitä se pitää sisällään. Verkkosivu on täysin frontend-painotteinen, jos sen toimintaan ei sisälly tarvetta tallentaa tietoja tietokantaan. (What's the Difference Between the Front-End and Back-End?)

Vue Native on frontend-termin alle asettuvaa toteutustyyliä sillä se on käyttöliittymän rakentamisen nopeuttamiseen kehitetty kirjasto.

2.2 Backend

Backend eli palvelinpuoli on tiedonsäilöntä painotteista tekniikkaa. Backend ohjelmoijat pitävät mielessään turvariskit, sivuston tietorakenteen ja sisällönhallintaan liittyvät seikat. Dynaamisiin ja alati muuttuviin sivujen ylläpitämiseen vaaditaan palvelinpuolen osaajia. Tietokannoissa säilytetään esimerkiksi tietoja käyttäjistä ja heidän asetuksistaan. (What's the Difference Between - - 2015.)

Node, Express ja MongoDB ovat termejä, jotka asettuvat backend-kokonaisuuden alle, koska niissä liikutetaan informaatiota tai luodaan keskusteluväyliä sovelluksen eri osien välille. Vue Native -kehityksen yhteydessä voidaan luoda testikanta, joka toimii lähiverkossa. Tämä helpottaa kehittämistä, kun oikeaa tilannetta voidaan soveltaa ilman lopullisia ratkaisuita.

2.3 HTML

HTML eli *hypertext markup language* on yksi internetin perusrakennuspalikoista. Sen suurin merkitys on antaa verkkosivuille merkitys ja rakenne. Sana *hypertext* viittaa verkkosivujen ydintoimintaan eli linkkien kautta navigoimiseen omien ja muiden luomusten välillä. HTML määrittelee useita eri kokonaisuuksia, joiden avulla verkkosivukokonaisuus voidaan muodostaa järkevästi. (HTML: Hypertext Markup Language 2020.)

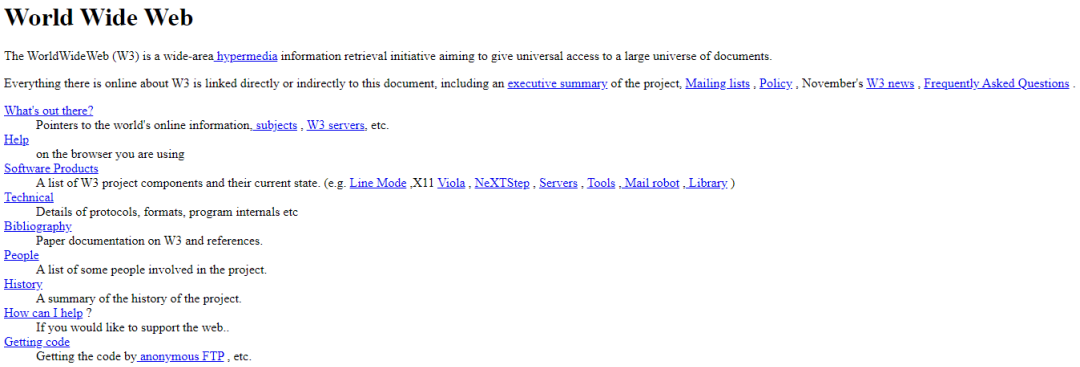
HTML on Vue Native -kehityksessä taustana *template*-mallin perusteella. Kehityksessä ei varsinaisesti esiinny samoja nimiä, mutta perusajatus on sama ja monen tasoiselle ohjelmoijalle tuttu.

2.4 CSS

CSS eli *cascading style sheets*, suomeksi porrastetut tyyliarkit, tekniikalla saadaan sivustot näyttämään paremmalta. Sitä oli mahdollista käyttää ensimmäistä kertaa vuonna 1996 Microsoftin Internet Explorerissa. (A Brief History of CSS 2016.)

HTML ja CSS ovat automaattisesti liitännäiset termit web-suunnittelijoille. Moni ehkä ajattelee, että ne luotiin yhdessä. Tämä ei kuitenkin ole totta, vaan kun HTML syntyi, sen ei ollut tarkoitus sisältää tyylimuokkaamisen mahdollisuuksia ollenkaan. (A Look Back at the History of CSS 2017.)

HTML ei mahdollista mullistavien visuaalisten ratkaisujen tekoa ja ilman CSS tekniikkaa jokainen verkkosivu näyttäisi samalta. Esimerkiksi tekstin sijaintia ja väriä ei voisi muuntaa, jos CSS:ää ei olisi ikinä kehitetty. Kuviossa 1 on kuvakaappaus ensimmäisestä maailmalle julkaistusta verkkosivusta.



World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

Kuvio 1. Ensimmäinen julkaistu verkkosivu (World Wide Web n.d.)

CSS on Vue Native -kehityksessä mukana, kun elementtien ulkoasua muutetaan. React toteutuksissa käytetään hieman eri kirjoitustyyliä näiden nimien suhteen, mutta muuten pitää tietää kaikki samat asiat. Tämä tekee Vue:sta helpommin lähestyttävän, koska kehittäjän ei tarvitse miettiä ja selvittää, miksi asiat eivät toimi.

2.5 JavaScript

JavaScript oli alun perin ainut keino saada verkkosivut tekemään muutakin kuin staattisia asioita. Vaikka nimessä on Java, ei *script*-kielellä ole mitään yhteyttä siihen, eikä se omaa samankaltaisia toimintatapoja. Mahdollisuus ajaa JavaScript-koodia serverillä on nostanut sen suosiota, ja sitä osaavien ohjelmoijien arvo on kasvanut. Enää ei tarvitse osata montaa erilaista kieltä eri käyttötarkoitukseen, mikä vähentää oppimisen määrää ja pienentää aikaa, jossa asiat saadaan täyttämään vaatimukset. (An Introduction to JavaScript 2020.)

Jokaisella selaimella, tai laitteella, joka pyörittää JavaScriptiä, on oma moottorinsa, jonka avulla kieli kääntyy sellaiseksi kieleksi, jota laite ymmärtää. Tällä käännöksellä taataan nopea suorituskyky. (An Introduction to JavaScript 2020.)

JavaScriptin toiminta perustuu olioihin ja prototyyppeihin. JavaScript ei asetu ainoastaan yhteen muottiin, vaan sillä voi kirjoittaa olio-ohjelmointia, imperatiivista- tai deklaratiivista- ohjelmointia. JavaScript on myös erittäin vapaamuotoista kirjoittaa. Kaikkia muuttujia ei tarvitse alustaa, eikä niitä tarvitse asettaa salatuiksi, suojatuiksi tai avoimiksi. Tämä on muista kielistä tuleville shokki. Tällaiset ominaisuudet tekevät siitä helposti lähestyttävän, mutta joskus ehkä liiankin vapaamuotoisen. (Introduction 2020.)

Jotta JavaScriptin maailmaan pääsee syventymään, on hyvä tietää, mitä ovat luokka ja olio. Luokka on eräänlainen pohjapiirustus, josta voidaan tehdä monenlaisia erilaisia, mutta silti samoja asioita omaavia asioita, kuten erilaisia taloja. Oliot siis perivät esimerkiksi värin, huoneiden määrän ja termostaatin lämpötilan luokalta, mutta niiden arvot voidaan määrittää erikseen. Näiden yhteistyöllä voidaan vähentää kirjoitettavan koodin määrää ja tehostaa tuotantoprosessia. (Python Object Oriented Programming n.d.)

Yksi JavaScriptin tärkeimmistä työkaluista on viitata verkkosivun dokumenttiolioon. Näiden viittausten pohjalta päästään käsiksi erilaisiin hyödyllisiin ominaisuuksiin kuten hiiren sijaintiin näytöllä. Dokumenttiolion kautta päästään käyttämään esimerkiksi *getElementsByClassName()*-metodia, jolla voidaan tietää tarkalleen mille elementille tehdään toiminnallisuuksia. Esimerkkinä tästä voi olla tilanne, jossa pitää tietää mitä tapahtuu “poista tavara ostoskorista” -napista. Silloin pitää tietää, mitä nappia on painettu, ettei synny virhetilannetta. (Document Object Model n.d.)

Joskus tarvitaan mahdollisuutta ajaa koodia samalla, kun toinen koodi suorittaa asioita, kuten tiedon noutamista palvelimelta. Tässä auttaa asynkroninen-koodi. Se on ollut suurimpia etuja Nodejs-maailmassa, joka ei kärsi prosessorin tukkimisesta, kuten vaikka PHP. Nykyään useat kielet ovat kuitenkin ottaneet mallia JavaScriptistä ja mahdollistaneet tämänkaltaisen koodin ajamisen kolmannen osapuolen lisäosien avulla. (Copes 2018a.)

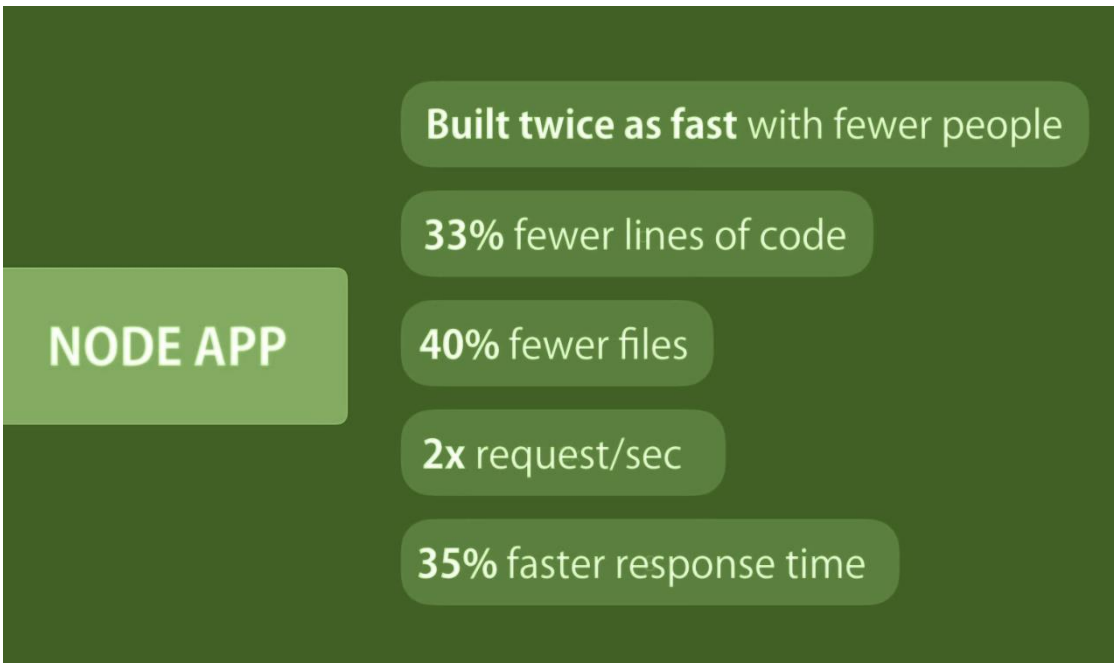
JavaScript on Vue Nativen peruspilari, joten sitä on osattava, jos haluaa hyödyntää Vue Native -kirjastoa kehityksessä.

2.6 Nodejs

Nodejs on avoimen lähdekoodin tuotos, jolla JavaScript-koodia voidaan ajaa erityyppisissä ympäristöissä. Avoimella lähdekoodilla tarkoitetaan sellaista teknistä toteutusta, jonka pohjimmaiset toimintatavat ovat kaikkien muokattavissa ja nähtävissä (What is open source? n.d.). Nodejs pohjautuu Google Chromen V8 -käännösmoottoriin, joka takaa huipputehokkaan toiminnallisuuden. Nodejs:n ei tarvitse jäädä odottamaan useiden eri kutsujen valmistumista, vaan se voi työstää useita asioita samanaikaisesti, tukkimatta kuitenkaan prosessoria. Tämä on esimerkki asynkronisesta toimintavasta. (A brief history of Node.js n.d.)

Nodejs:n kasvua auttavat myös runsaat käyttäjien luomat kirjastot, joilla puuttuvia toiminnallisuuksia voidaan korvata erittäin nopealla tahdilla. (Express/Node introduction 2020.)

Opinnäytetyössä oli tarvittavaa saada backend-palvelin rakennettua, jotta sovellusta pystyttiin testaamaan oikeantyylisissä tilanteissa. Nodejs on siihen täydellinen, varsinkin kun mukana oli MongoDB. Kuviossa 2 on tietoja PayPal:n havaitsemista hyödyistä, kun he toteuttivat projektin käyttäen Nodejs-teknologiaa. Kuvion sisältö vertaa Nodejs projektia Java ja Spring -pohjaiseen toteutukseen. Nodejs-toteutus valmistui kaksi kertaa nopeammin vähemmällä määrällä ihmisiä, kokonaisuudessaan koodirivejä oli 33% vähemmän, tiedostoja oli 40% vähemmän, pyynnöt nousivat kaksinkertaisiksi sekunnin aikana ja responssiaika kasvoi 35%.



Kuvio 2 PayPalin luoman projektin Node hyödyt (Hamedani 2018, muokattu.)

2.7 Npm

Node package manager eli *npm* mahdollistaa helpon lisäosien asentamisen Node ympäristöihin. Npm asentaa tarvittavat tiedostot ja kirjastot *node_modules* kansioon. Tämän kansion koko kasvaa kuitenkin melko suureksi, joten sen liikuttaminen internetissä kestäisi pitkään. Tässä asiassa auttaa *package.json*-tiedosto, jossa on tieto kaikista halutuista paketeista ja niiden versioista. Jos projektista löytyy package.json-tiedosto ja käyttäjä ajaa komennon *npm install*, kaikki toimintaan vaaditut paketit saadaan ladattua helposti yhdellä komennolla. (What is npm? 2011.)

Npm on jokaisessa projektissa tarvittava väline, jotta tiedostoista saadaan koostettua käynnistävä kokonaisuus. Vastaavasti voidaan hyödyntää uudempaa tulokasta nimeltä *yarn*.

2.8 Express.js

Express on Node-kirjastoviitekehys eli se laajentaa ja tehostaa Noden toimintaa. Expressin avulla voidaan luoda polkuja, joiden kautta tietokannasta haetaan tietoa ja lähetetään käsittelypyyntöjä. Expressin käyttäjät ovat laajentaneet koodia ja täten saaneet käyttöön lähes kaikki halutut toimintatapaukset. Erilaisia ratkaisuita löytyy kuitenkin liiaksikin, joten voi olla vaikea löytää se juuri omaan käyttötarkoitukseen soveltuva keino. (Express/Node introduction 2020.)

Express on äärimmäisen vapaasti muokattavissa ja tarkkoja rajauksia ei ole noudatettava. Ohjelmoijalla on vapaat kädet päättää kansiorakenteesta ja Express-kokonaisuuteen voi halutessaan liittää melkein mitä toiminnallisuutta tahansa. (Express/Node introduction 2020.)

Vue Native -kehityksessä Express toimii välikätenä sovelluksen ja backendin välissä. Sen avulla voidaan muuttaa tietoja, hakea tietoja ja lisätä tietoja tietokantaan.

2.9 JSON

JSON tulee sanoista *JavaScript object notation* ja se on hyväksi todettu malli järjestää tietoa. JSONia voi ajatella listana olioita. Koodissa tällaisiin olioihin voidaan viitata käyttämällä *tietue.kentännimi* -tyyliä. JSONiin voidaan myös sisällyttää taulukoita, joka laajentaa hyödyllisyyttä entisestään. JSON-olion sisällä voi myös olla toisia olioita, joilla tiedon järjestelmällisyys saadaan maksimoitua. (JSON: What It Is, How It Works, & How to Use It n.d.)

JSON on ollut äärimmäisen tärkeä osa nykymaailman verkkosivujen kehitystä. Ennen verkkosivuja jouduttiin lataamaan kokonaan uusiksi aina kun jokin tieto päivittyi,

mutta JSON-tiedostomuoto ja API-rajapinnat mahdollistivat, että tietoa voidaan ladata ja päivittää ilman sivujen uudelleenlataamista. Tämä on tehostanut käyttöä ja tehnyt selaamisesta paljon nopeampaa. (Freeman 2019.)

JSON-mallin toiminta on Vue Native -kehityksessä tärkeä ymmärtää, jos tekee rajapinta kyselyitä ja haluaa tietää mitä tietoa backend-toimitti sovellukseen.

2.10 API

API eli englanniksi *application programming interface*, mahdollistaa ohjelmiston viestinnän rajapintojen kanssa. Rajapinta on ohjelmiston ja tietovaraston välimaasto, jossa eri puolet viestivät toistensa kanssa. API-kutsuja on erilaisia, joista useimmat liittyvät tiedon hakemiseen ja muokkaamiseen. Tietoturva on näissä kuitenkin sen verran suuri riski, joten kelle tahansa ei pitäisi antaa oikeuksia urkkia tietoa tai muokata sitä. Tähän hyödynnetään API-avaimien luontia. Niillä varmistetaan, että käyttäjällä, joka kutsuu rajapintaa, on oikeus tehdä asioita. API:t mahdollistavat yritysten välisen datan jakamisen, mikä taas nopeuttaa ohjelmistojen kasvua. (Tasanen 2019.)

Yksi arkipäivän esimerkeistä API:sta ovat palveluiden sisäänkirjautumismahdollisuudet. Useat sivut tarjoavat Google, Facebook ja Github -kirjautumista heidän API:n avulla, joka säästää vaivaa ja aikaa. Verkkosivu lähettää API-kyselyn esimerkiksi Googlen-serverille, Google vastaa, että kyllä tällainen käyttäjä löytyy, ja luo sen jälkeen avaimen, jonka avulla käyttäjä voidaan tunnistaa luotettavaksi. APIen avulla yritykset voivat luoda verkkopolkuja, joissa ei liiku muuta kuin raakaa tekstipohjaista dataa, ja nopeuttaa näin sivustojen toimintaa. (Gazarov 2019.)

Vue Native -sovellukset vaativat yleensä rajapintoja, jotta informaatiota saadaan sovellukseen käsiteltäväksi. Opinnäytetyössä luotiin rajapinta, jonka kautta eri tietoja haettiin ja muokattiin.

2.11 Tietokanta

Tietokannat ovat systemaattiseen tiedonsäilömiseen käytettyjä tietopankkeja. Tietokannoista voidaan noutaa tietoa, muokata sitä tai lisätä uutta tietoa. Tietokanta ihmisistä voisi sisältää esimerkiksi osoitteen, puhelinnumeron, pituuden, painon ja iän. (What is Database? What is SQL?)

Tietokantojen muokkaamiseen löytyy eri systeemeitä. Hierarkkinen tyyppi perustuu lapsi-vanhempi yhteyksiin. Verkkotyyppinen ratkaisu perustuu monesta tietueesta moneen yhteyksiin. Relationaalinen-malli perustuu muodostettuihin tauluihin ja niiden suhteellisuuteen. Uusimpana tulokkaana on syntynyt myös oliomallin relaatiokantoja. (What is Database? What is SQL? n.d.)

2.11.1 SQL

Structured Query Language, lyhyesti SQL, mahdollistaa kommunikoinnin tietokantaan. Käyttö perustuu suurimmaksi osaksi seuraaviin sanoihin: *select*, *insert*, *update*, *delete*, *create*, ja *drop* (What is SQL?). Selectillä voidaan hakea tietueita tietokannasta (Selecting Data n.d.). Insertillä voidaan lisätä uusia tietueita (Inserting into a Table n.d.). Updatella voidaan päivittää jo olemassa olevaa tietoa. (Updating Records n.d.) Deletellä voidaan poistaa tietoja kannasta (Deleting Records n.d.). Creatella luoda uusia tauluja, joihin saadaan taas uuttaa tietoa (Creating Tables n.d.). Dropilla voidaan pudottaa ja siten hävittää tauluja, mutta usein taulujen tulee olla tyhjiä, että tämä toiminto on mahdollinen (Drop a Table n.d.).

2.11.2 NoSQL

NoSQL tulee sanoista *not only sql*. NoSQL on noussut suosiossa sen skaalautuvuuden ja joustavuuden ansiosta. Myös tehokkuudessa ja tiedon saavutettavuudessa on paljon hyviä edistyksiä. NoSQL loistaa datan varastoinnissa, jolle ei ole määritetty tarkkaa rakennetta. Key-value, avain-arvopari, varastot ovat suosittu NoSQL-tyyli. JSON

kuuluu tällaisiin varastotyyppeihin. Tiedostovarastot ovat samantyyppisiä, mutta jokaisella tietoryppäällä on oma avaimensa, joka helpottaa tiedonhaussa. *Wide-column* -varastot matkivat normaalimpaan sql-mallia. Viimeisimpänä Graph-varastot käyttävät hyväkseen viereisten tietoryhmien relaatiosuhteita toiminnassaan. (NoSQL Databases Explained n.d.)

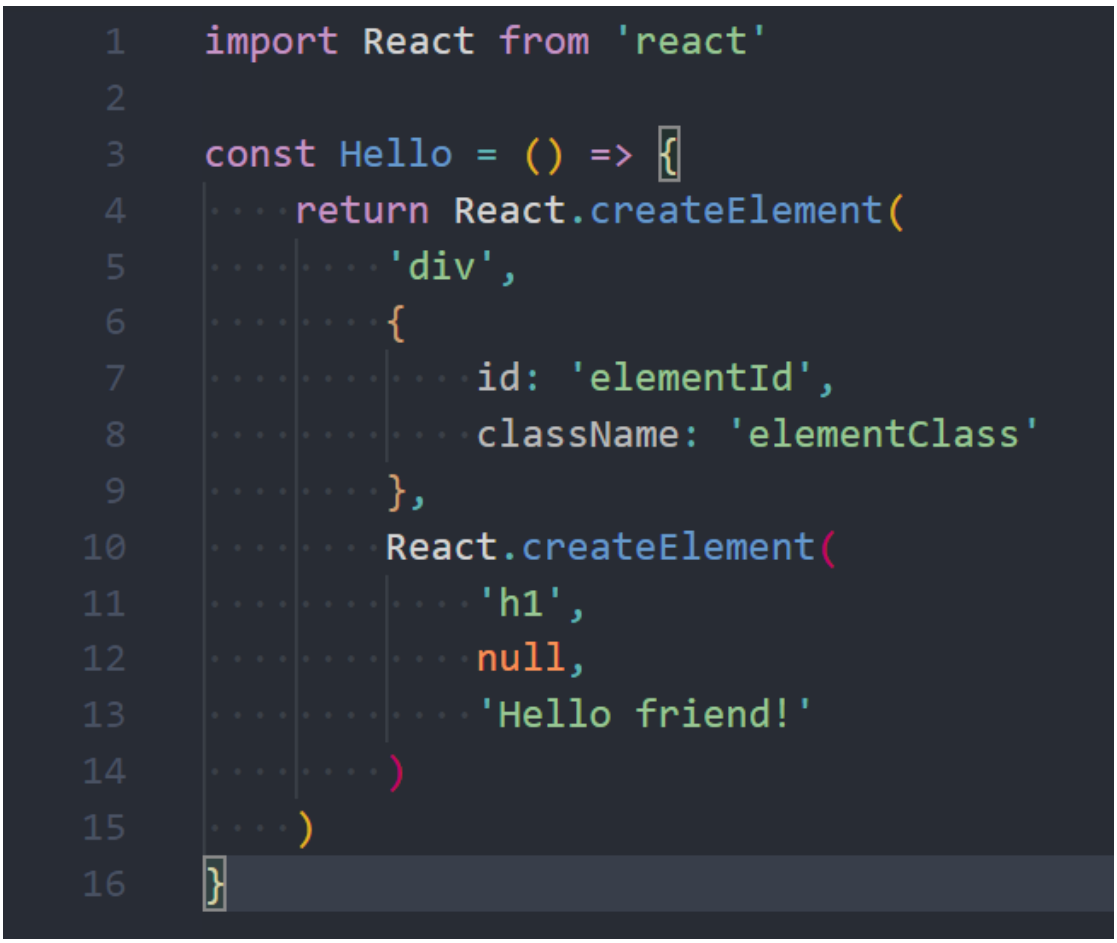
2.11.3 MongoDB

MongoDB on suosituimpia NoSQL-tietokantoja sen suurten tietomäärien tallennuskykyjen ansiosta. Mongossa käytetään dokumenttimallia, joka tekee siitä erittäin joustavan ja käyttäjäystävällisen. Dokumenttimallin rakenne vastaa ohjelmointikielistä tuttuja rakenteita. Mongossa ei ole tarpeellista määritellä liikoja etukäteen, vaan asioita voidaan muokata lennosta. Se on yleensä tarpeellista ennen pitkää. Yleensä projektin aikana tuleekin yllätyksiä ja huomataan, että asiat pitääkin tehdä toisin. (What is MongoDB? n.d.)

MongoDB oli käytössä Vue Native -sovelluksessa, jotta toimintaa pystyttiin demonstroimaan oikeantyyppisissä tilanteissa. Se valittiin skaalautuvuuden ja helpon muokattavuuden toivossa.

2.12 JSX

JSX on lyhenne sanoista *JavaScript XML*. Toisin sanoen se mahdollistaa HTML-tyylisen kirjoitustavan JavaScript-koodin sisällä. JSX kääntyy puhtaaksi JavaScriptiksi selaimissa. Tälle tavalle altistuu React-koodia kirjoittaessa, mutta sen käyttäminen ei ole välttämätöntä siinäkään. Kuitenkin, jos luotavia elementtejä pitää luoda useita, kestää niiden kirjoittaminen ilman JSX:ää kauemmin. Kuviossa 3 toimitaan ilman JSX:n apuja ja kuviossa 4 nähdään selkeämpi JSX-toteutus. (ReactJS Tutorial - 8 – JSX 2018.)



```
import React from 'react'

const Hello = () => {
    return React.createElement(
        'div',
        {
            id: 'elementId',
            className: 'elementClass'
        },
        React.createElement(
            'h1',
            null,
            'Hello friend!'
        )
    )
}
```

Kuvio 3. Elementin luonti ilman JSX-kieltä

Kuvio 4. Elementin luonti käyttäen JSX-kieltä

HTML:ssä on totuttu käyttämään sanaa *class* kun elementille pitää antaa kutsumanimi tyylejä tai tunnistamista varten. JSX:ssä tämä ei kuitenkin onnistu, koska *class* on varattu nimi JavaScript-ympäristössä. Tästä johtuu hieman erilainen nimeämistapa *className*. Myös funktioiden nimeäminen on erilaista. Reactissa nimeäminen perustuu *camelCase*-nimiseen nimeämistapaan, jossa funktioiden ja muuttujien sanan ensimmäinen kirjain on aina pienellä, mutta seuraavat sanat alkavat isolla. Tulevaisuudessa nämä nimeämistavat saattavat muuttua takaisin vastaamaan yleisimmin nähtyjä malleja. (ReactJS Tutorial - 8 – JSX 2018.)

JSX-tyyliä kirjoittaa sanotaan joissakin piireissä syntaattiseksi sokeriksi, joka tarkoittaa maanläheisemmin yksinkertaisempaa ja helpommin ymmärrettävää tapaa kirjoittaa jokin koodiosuus. (Baig 2018.)

2.13 React.js

React.js on Facebookin ylläpitämä JavaScript-kirjasto, joka on mullistanut verkkokehitystä viime vuosina. Ydinajatuksena Reactissa ovat komponentit. Komponentit ovat

uudelleenkäytettäviä paloja, joita voidaan käyttää HTML-elementtien tapaan, mutta niiden toiminnallisuus on kehittäjän itsensä käsissä (Web Components 2019). Komponenttien toiminta perustuu *state*-nimisen kokonaisuuden muokkaamiseen. State mahdollistaa tietojen keskittämisen yhteen paikkaan. Tähän tilaan viittaamisella vältytään turhalta kopioinnilta. (Introduction to React n.d.)

Komponenttien toimintaa tehostaa *Virtual DOM*, jonka React esitteli suurelle yleisölle suosion kasvaessa. Virtual DOM elää verkkosivun tai sovelluksen pohjapiirros-DOMin eli *Document Object Model*:n päällä. Tämä virtuaalinen dokumenttioliomalli tarkkailee muutoksia ja päivittää vain ne tietyt osat, joihin näillä muutoksilla on vaikutusta. Huolimatta tällaisesta raskaasta toiminnallisuudesta, React on onnistunut pitämään suorituskyvyn korkeana. Se on kuitenkin kärsinyt pakkausten koossa, joka rajoittaa sen laajenemista joissakin määrin, ainakin sellaisiin paikkoihin missä internetnopeuksissa on vielä parannettavaa. (Introduction to React n.d.)

Reactissa on kaksi tapaa alustaa komponentteja. Alkuperäinen tyyli perustuu luokkien luomiseen ja uudempi vuonna 2019 julkistettu funktionaaliseen toteutustapaan nojaava *Hooks*. (McGinnis 2019.)

Kaksi eniten esiintyvää sanaa React-kehityksessä ovat *state* ja *props*. State säilyttää tietoa sisällään ja siitä voidaan jakaa osia, tai *state* kokonaan, propsien kautta. Tällä tavalla tietoa voidaan säilöä ainoastaan yhdessä paikassa, eikä sitä tarvitse monistaa jokaisessa osassa sovellusta erikseen. *Propseja* ei pidä ikinä yrittää muokata suoraan, ilman, että *state* muuttuu. *Statea* voi muokata suoraan, mutta se on vastoin React-kehittäjien haluttua toimintatapaa. Jos *statea* halutaan muokata, se tulee tehdä *setState()*-funktion kautta. (Component State n.d.)

Stateista ja propseista käytännön esimerkkinä voi kuvitella tilannetta, jossa on sovellus, joka näyttää säätietoja. Koko viikon säätieto-komponentti voi antaa propseina maanantai-komponentilla tiedot maanantain säätiedosta, joka sitten tulostaa ne

erikseen. Näin vältytään tiedon turhalta monistamiselta, eikä maanantai-komponentin tarvitse olla tietoinen tiistaina tapahtuvista asioista.

Jos komponentti on luokkapohjainen, siihen voidaan liittää erilaisia *lifecycleja* eli komponentin elinkaaren vaiheissa voidaan ajaa eri funktioita. Metodi *componentDidMount()* ajetaan, kun komponentti on alustettu. Tämä on eniten käytetty elinkaarimetodi. Toinen samankaltainen ja nimessäänkin toimintoa kuvaava on *componentWillUnmount()*. Tämä ajetaan, kun komponentti ollaan tuhoamassa ja se poistetaan muistista. (State and Lifecycle n.d.)

Reactissa on kahdenlaista tapaa luoda komponentteja. Funktionaalinen malli on niin sanotusti ”tyhmä”, koska se ei voi sisältää state-ominaisuutta. Se vastaa yleistä JavaScript-funktioiden mallia. (Functional Components vs Class Components n.d.)

Vuonna 2015 julkaistussa ECMAscript-standardissa esiteltiin nopeampi tapa luoda funktioita. Viimeksi JavaScript sai uusia merkittäviä muutoksia vuonna 2009. Nuolifunktiot, englanniksi *arrow functions*, poisti tarpeen määritellä *function* aina kun luodaan uusia funktioita. Java, C# ja muut omaavat samankaltaisia menetelmiä. Yksi merkittävimmistä muutoksista tuli *this*-termin viittaamisen muodossa. Ennen ES6 standardia, *this* piti erikseen asettaa vastaamaan oikeaa funktiota. Jos se jätettiin tekemättä, kehittäjät kohtasivat vaikeasti kartoitettavia ongelmia. (ECMAScript 6 n.d.)

ES5 ja ES6 -funktioiden luettavuutta voi verrata kuviosta 5.

```
3   function timesTwo(params) {
4     return params * 2;
5   }
6
7   const timesTwo = (params) => params * 2;
8
9   timesTwo(4); // 8
```

Kuvio 5 ES5 funktion ja ES6 funktion eroja (Lee 2018)

Class based, eli luokasta periytyvät, komponentit taas ovat suurimman työn tekeviä osia Reactissa. Nämä komponentit perivät React-komponentista tarvittavat ominaisuudet. Samalla nämä komponentit saavat käyttöönsä aiemmin mainitut *lifecycle*-metodit. Kuviossa 6 ylempänä on funktionaalinen komponentti ja alla class-pohjainen.

Kuvio 6 Funktionaalisen ja class -komponentin eroja (Jöch 2018)

2.14 Vue.js

Vue.js on Evan You:n kehittämä *JavaScript-framework*. Evan You kehitti Vue-kirjastoa samalla kun hän työskenteli Googlella. Luomisen taustalla oli halu tehostaa sovelluksia. Vue on helpommin lähestyttävä, sillä se tekee taustalla paljon asioita, joita React:ssa kehittäjän pitää itse huomioida. Myös Vue hyödyntää toiminnassaan Virtual DOM:ia. Vuen tukena ei ole mitään suurta korporaatiota, joka estäisi kehitystä ja

kasvua. Yhteisö rahoittaa ja vie tekniikkaa eteenpäin omassa tahdissa. Vue suosii *template*-systeemiä, eli mikä tahansa HMTL-runko on käytettävissä. (Copes 2018c.)

Vue-komponenteissa on kolme tärkeää osaa: tuloste eli template, toiminnallisuus ja tyylit. Ylimpänä toimivassa tulosteosassa määritellään mitä sovelluksen käyttäjä näkee. Toiminnallisuus osaan luodaan nimensä mukaisesti komponentin ydintoiminnot. Tyyli osuudessa voidaan antaa tyylejä ainoastaan halutulle komponentille. Tämä on hyödyllistä, etteivät tyylit sekoitu komponenttien kesken. (Traversy 2019a, 4:34.)

Kuvioissa 7 on kuvakaappaus Vue *template*-koodista. Sivulle tulostetaan laskuri, jota voidaan kasvattaa ja vähentää napeilla. Vue:ssa on omia käyttöä helpottavia lyhenteitä kuten *@click*, joka lähettää tiedon, että siihen annettu funktio ajetaan, kun käyttäjä painaa nappia. Omituinen *{{ counter }}* -kokonaisuus mahdollistaa muuttuvan tiedon hakemisen komponentin tietovarastosta. Ilman tätä olisi hankala saada dynaamisuutta aikaan.

```
<template>
  <div id="app">
    <div id="content">
      <h1 class="header">Counter currently {{counter}}</h1>
        <div class="buttons">
          <button @click="increaseCounter" class="btn-style">+</button>
          <button @click="decreaseCounter" class="btn-style">-</button>
        </div>
    </div>
  </div>
</template>
```

Kuvio 7. Esimerkki Vue template-koodista

Kuviossa 8 määritellään komponentin ydintoiminnot. Vue-komponenteissa jokainen funktio määritellään *methods*-nimisen lohkon sisään. Tämä selkeyttää koostamista. Toinen metodi vähentää komponentin tietovarastossa olevaa *counter*-arvoa yhdellä ja toinen lisää sitä. Tässä on hyvä verrata React-toimintatapaan, jossa pitäisi hyödyntää *setState()*-funktiota, mutta Vue:ssa sen kaltaisen toiminnan poisjääminen selkeyttää tekemistä. Vue:ssa on Reactistakin tuttu *componentWillMount()* tyyppinen ratkaisu. Vue:ssa se on kuitenkin vain nimellä *mounted()*. Kuvion koodissa *counter*-arvo asetetaan lukuun 100, kun komponentti on alustettu.

```
<script>

  export default {

    data() {
      return {
        counter: 0
      }
    },

    methods: {

      increaseCounter() {
        this.counter += 1
      },

      decreaseCounter() {
        this.counter -= 1
      },

    },

    mounted() {
      this.counter = 100
    }

  }

</script>
```

Kuvio 8. Esimerkki Vue script-osiosta

Kuviossa 9 on esimerkkiä tyyleistä. Tärkeimpänä huomiona tässä on sana *scoped*. Tämä rajaa tyylit ainoastaan kyseisen komponentin käyttöön.

```
<style scoped>
  #app {
    display: flex;
    justify-content: center;
    height: 100vh;
    width: 100vw;
  }
  #content {
    align-items: center;
  }
  .header {
    font-size: 2em;
    text-align: center;
  }
  .buttons {
    align-self: center;
  }
  .btn-style {
    background-color: rgb(131, 131, 131);
    border: none;
    margin: 1em;
    height: 64px;
    width: 64px;
    border-radius: 50%;
    color: white;
    font-size: 2em;
  }
</style>
```

Kuvio 9. Esimerkki Vue style-osiosta

Mounted-funktion lisäksi Vue:ssa esiintyy myös seuraavat: *beforeCreate*, *created*, *beforeMount*, *beforeUpdated*, *updated*, *beforeDestroy* ja *destroyed*. Vähemmän törmättyihin kuuluvat *activated* ja *deactivated*. (Bemenderfer 2017.)

Näiden kaikkien toiminnasta saa aika hyvän kuvan jo nimistä, mutta nyanssit ovat melko hankala ymmärtää sataprosenttisesti. Kokeilemalla löytää omiin käyttötarpeisiin hyödyllisimmät.

2.15 React ja Vue vuonna 2020

Reactin ja Vuen taistelu jatkuu varmuudella myös vuonna 2020. Siksi on hyvä tietää, kumpi kannattaa valita, jos suunnittelee projektia.

Vue on helpompi oppia HTML-template syntaksin ansiosta. Sen koodimuodot ovat helposti ymmärrettäviä ja luettavampia kuin Reactissa. Vue omaa myös hyvän dokumentaation. React on edellä tarjottujen lisäosien muodossa ja myös siinä, että se on laajemmassa käytössä jo olemassa olevissa projekteissa. Vue on helppo lisätä mukaan projektiin pienenä kirjastona tuomaan esimerkiksi interaktiivisuutta. Vue on täten erittäin edullinen pieniin ja keskikokoisiin projekteihin, koska sillä on helppo luoda ideat konkreettisiksi sisällöiksi. (Nowak 2019.)

Vue hyödyntää template-muotoa, koska se on tutumpaa monille kehittäjille. Se on myös luonnollisempaa lukea ja kirjoittaa. Vue on hyvä valinta myös suunnittelijoita silmällä pitäen, koska heidän on helpompia osallistua kehitykseen ymmärrettävyyden ansiosta. (Comparison with Other Frameworks n.d.)

React luotiin isoille web-projekteille. Pienissä sovelluksissa se voi tarjota liikaakin. JSX tarjoaa kaikki JavaScript hyödyt mukanaan. Jos React-sovelluksessa haluaa hyödyntää reititystä tai tilanhallintaa, pitää valita ulkopuolisten tarjoajien luomuksista sopivin. (Nowak 2019.)

Kuviossa 10 nähdään, miten React-komponenteissa voidaan toimia tyylien kanssa. Se eroaa normaalista tavasta ja tyyli tunnetaan nimellä *CSS-in-JS*. (Styling and CSS n.d.)

```
44  const StylingInReact = () => {
45    const wrapper = {
46      display: "flex",
47      flexDirection: "column",
48      alignItems: "center",
49      justifyContent: "center",
50      width: "100%",
51      color: "#eee",
52      backgroundColor: "red",
53    };
54
55    return (
56      <div style={wrapper}>
57        <h1>Hello Styling</h1>
58      </div>
59    );
60  };
61
62  export default StylingInReact;
```

Kuvio 10 React-komponentin tyylien asettaminen

2.16 Native

React Native ja Vue Native omaavat sanan *Native*, koska näiden avulla voidaan kirjoittaa koodia yhdellä kielellä ja kirjastot itsessään huolehtivat kääntämisestä

Android ja iOS versioiksi. Tässä tapauksessa säästetään paljon aikaa ja rahaa. Kehittäjät voivat kehittyä ja maksimoida osaamistaan yhden kielen opiskeluun. (Traversy 2020, 1:15.)

Ennen vuotta 2015 oli vaikea miettiä, miten saman koodin pohjalta saataisiin toimiva sovellus usealle alustalle. Alustoja oli liikaa ja käyttäjät asettuivat liian laajalle alueelle. iOS ja Android -alustojen kautta saavutetaan nykyään 99% käyttäjistä. Microsoft hankki oikeudet Xamariniin, avaten samalla lähdekoodin avoimeen käyttöön vuonna 2016. Tietoisuus käyttäjien asettumisesta ja Microsoftin edesottamukset nopeuttivat native-kehityksen etenemistä. Ennen puhtaasti native-ratkaisuja puhuttiin yleensä hybrid-malleista, jotka hyödynsivät HTML, JavaScript ja CSS -tekniikoita, mutta ne päällystettiin kuorella, jonka kautta saatiin sovellus, jonka sai Google tai Apple -storeen. Näissä ratkaisuissa suorituskyky ei yltänyt samalla tasolle. (Maggini 2019.)

2.17 Vuex

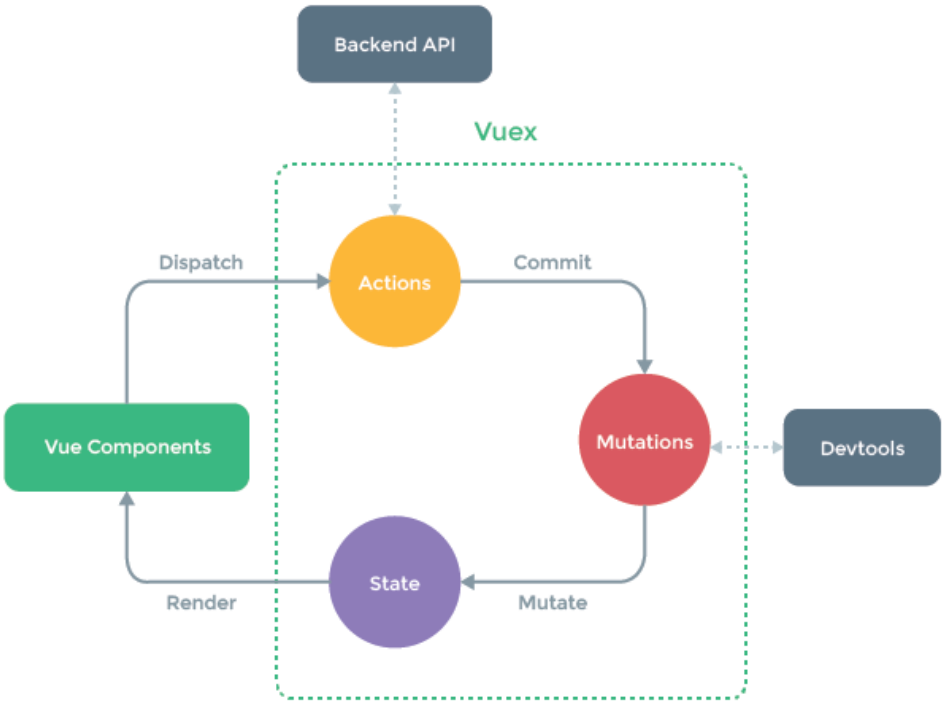
Sovellukset kasvat joskus isoiksi, jolloin on vaikea saada tarvittava data kaikille komponenteille käyttöön. Propsien liikuttelu muuttuu erittäin työlääksi, kun sovellukseen tulee lisää tasoja ja se alkaa syventyä. Näissä tilanteissa apuun tulee *state management* eli tilanhallinta-kirjastot kuten Vuex. Reactin puolella tunnettu vastaava järjestelmä on Redux. Vuex perustuu periaatteeseen nimeltä *single source of truth*. (Traversy 2019b.)

Single source of truth -periaate tarkoittaa, että kaikki sovelluksen data on tallessa store-tiedostossa sijaitevassa state-puussa valmiina käyttöön ja jaettavaksi. Näin varmistetaan helpompi ongelmien haravointi ja helpotetaan ymmärtämistä, kun datamäärät kasvavat ja mukaan tulee uusia kehittäjiä. (Three Principles n.d.)

Vuex-terminologiassa esiintyy sanoja kuten *mutation*, *action*, *store*, *modules* ja *getters*. Nämä kaikki ovat osa Vuex-filosofiaa, jossa statea ei pitäisi ikinä muokata suoraan, vaan sen pitää tapahtua *mutation*-osan kautta.

Vue Native -sovelluksessa ei välttämättä tarvitse ottaa käyttöön Vuex-teknologiaa. Opinnäytetyön sovelluksessa oli kuitenkin tarpeellista saada data pääasiallisesti yhteen sijaintiin, jotta projektilistaan saatiin tarvittavat tiedot. Varsinkin prosenttiosuus oli tarvittavaa saada näkyviin.

Kuviossa 11 on esitetty mallia Vuex:n asettumisesta sovellukseen.



Kuvio 11. Vuex asettuminen Vue-ympäristöön (What is Vuex? n.d.)

2.18 VsCode

VsCode, kokonaisuudessaan *Visual Studio Code*, on tällä hetkellä suosituin koodi tekstieditori. Sen takana on Microsoft, mutta sen kehitys on avoimeen lähdekoodiin perustuvaa. VsCodessa on ensi asennuksesta lähtien paljon hyvää, mutta sen toimivuutta voidaan entisestään laajentaa lisäosien avulla. VsCodesta löytyy myös terminaali ja Git-ominaisuudet, jotka ovat jokaisen kehittäjän tuttuja ystäviä. Tämä ohjelma nopeuttaa työskentelyä huomattavasti. (Copes 2018b.)

VsCode tarjoaa paljon lisäosia ja se on totta myös Vue Nativen kohdalla. Vue lisäosien avulla saa käyttöön *intellisense*-ominaisuuden, joka ehdottaa virheistä hyvissä ajoin. Editori myös ehdottaa vaihtoehtoja kirjoitetun perusteella. Node ja Express koodiinkin saatiin samalla tavalla ilmoituksia.

2.19 Android Studio

Android Studio on Googlen tukema Android-käyttöjärjestelmälle tarkoitettu Java-pohjainen sovelluskehitysympäristö. Android Studiossa on paljon osia, joilla voidaan luoda apk-tiedostoja, jotka ovat Android järjestelmien sovellus asennustiedostoja. Tästä toiminnosta työkalussa vastaa Gradle. Android Studio mahdollistaa myös sovellusten ajamisen emulaattorien kautta. (Friesen 2020.)

Emulaattorien avulla voidaan ajaa jotain toista tietokonejärjestelmä toisen järjestelmän kautta. Esimerkiksi Mac koneen sisällä voidaan ajaa Windows-ympäristöä. (Wells 2019.)

Android Studio on tarpeellinen väline Vue Native -kehityksessä, jos ei omista fyysistä Android-laitetta. Android Studio tarjoaa hyvän emulaattorin, jonka kautta voi testata oman sovelluksensa toimivuutta.

2.20 Expo

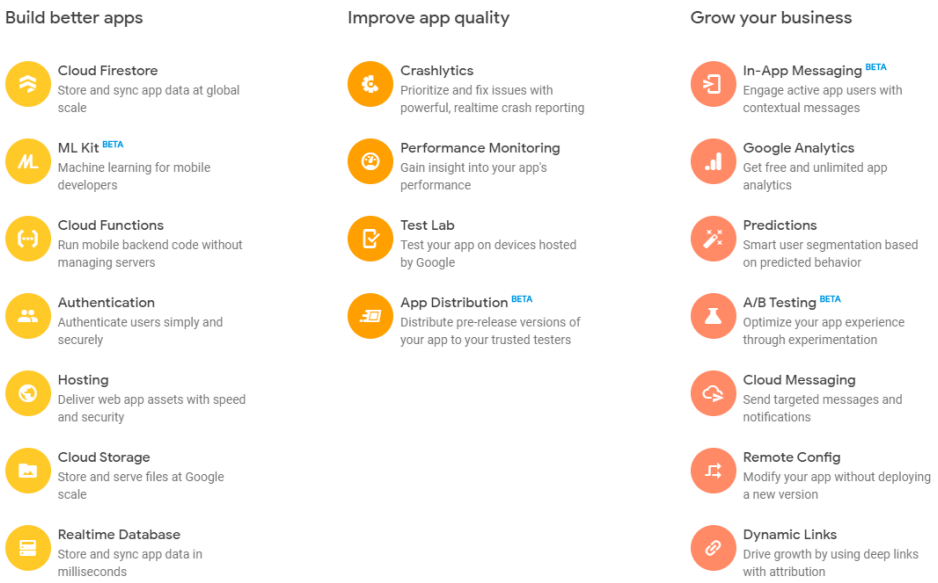
Expo on suosittu Native-kehityksessä käytetty apuväline. Se poistaa useita esteitä, joihin kehittäjä saattaa törmätä mobiilisovellusten luonnissa.

Expossa on kuitenkin rajoituksensa. Jos haluaa kirjoittaa osia Applen Swift -kielellä tai Androidiin Javalla sekä Kotlinilla, niitä osia ei saa käyttöön Expo-sovellukseen. Expo-sovellukset eivät pysty toistamaan ääntä, jos sovelluksesta poistutaan. Spotifyn tyylisiä soittimia ei siis voi tällä hetkellä luoda. Myös push-notifikaatioiden kanssa ei ole paljon valinnanvaraa ja pitää tyytyä Expon tarjoamiin ominaisuuksiin. Jos sovelluksesta haluaa apk- ja ipa-tiedostoja ne pitää ajaa Expon sovelluskasaajan kautta, joka vaatii usein jonottamista. Tämä taas vie aikaa ja mahdollisesti rahaa. (Expo Limitations n.d.)

2.21 Firebase

Firebase on Googlen tarjoama kokoelma pilvipalveluita, joihin kuuluvat esimerkiksi authentikaatio työkalut ja pilvitietokantojen luonti. Yleensä tämänkaltaiset systeemit joudutaan luomaan alusta alkaen jokaiseen sovellukseen. Firebase-kokoelma auttaa kehittäjiä säästämään aikaa ja mahdollistaa turvallisen vaihtoehdon. (Stevenson 2018.)

Alapuolen kuviossa 12 on esitetty tämänhetkiset erilaiset mahdollisuudet.



Kuvio 12. Kuvakaappaus Firebase tarjoamista hyödyistä (Google Firebase n.d.)

Tällä hetkellä Firebase-SDK on suunnattu enemmän Android ja iOS -alustoille (What is Firebase? The complete story, abridged.). SDK eli *software development kit* sisältää erilaisia työkaluja, kirjastoja, dokumentaatiota ja koodiesimerkkejä helpottamaan ohjelmoijien työtehokkuutta (Sandoval 2018).

Opinnäytetyön sovellukseen otettiin käyttöön Google-autentikaatio.

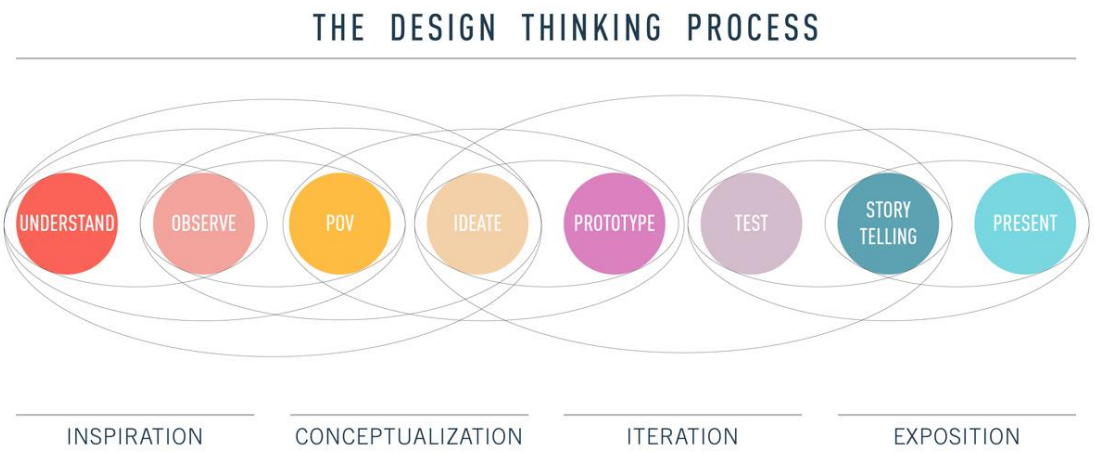
2.22 Käytettävyys

User Experience eli UX-termi esiintyy silloin kun mietitään käyttäjän kokemuksia heidän käyttäessään jotain laitetta tai palvelua.

UX sekoitetaan helposti samantyyliseen sanaan UI, joka taas tulee sanoista *user interface*. UX keskittyy käyttäjäkokemukseen ja sen mielekkyyteen. Nämä molemmat ovat

kuitenkin suuressa roolissa sovelluksen laadunvarmistamisessa. UX:llä on pitkä historia, joka on vasta teollisuusvuosina alkanut kehittyä omaksi osa-alueekseen. 1990-vuosikymmenen alussa Don Norman sai ensimmäisen UX-suunnittelija tittelillä nimitetyn työpaikan Apple:lla. Viime vuosina nimikkeitä on alkanut syntyä yhä tarkennettuina eri tehtäville erikseen. (Stevens 2019.)

UX-suunnittelijan tehtävänä on luoda mahdollisimman helposti käytettäviä laitteita, palveluita ja teknologioita. Alla esitetyssä kuviossa 13 on yksi malli kehitysprosessi vaiheista. (Stevens 2019.)



Kuvio 13. Suunniteluprosessin vaiheita (Stevens 2019)

Peter Morville kehitti idean, jonka hän esitti hunajakennokuviona. Kuvio sisältää seitsemän eri aluetta, jotka ovat osana suunnittelua. *Usable* kuvaa miten käytettävä ja

intuitiivinen jokin asia on. *Useful* viittaa hyödyllisyyteen ja tarpeellisuuteen. *Desirable* syntyy käyttäjän tuntemusten kautta sekä brändin ja värien avulla. *Accessible* viittaa sellaisten ihmisten huomioitiin, joilla on joitain rajoitteita kuten värisokeus. *Credible* viittaa luotettavuuteen – käyttäjien pitää luottaa, että esitetty tieto on uskottavaa. Viimeisimpänä *findable* koostaa ajatuksen tiedon löydettävyydestä navigoinnin tai vastaavan avulla. Kaikki nämä yhdistyvät *valuable* kokonaisuuteen, josta syntyy tuotteen arvo. Kuviosta 14 voidaan nähdä kokonaisuus täydellisyydessään. (User Experience Basics n.d.)

Kuvio 14. Käytettävyyssuunnittelun hunajakenno (User Experience Basics n.d.)

2.23 Käyttöliittymä

Aiemmin mainittu *user interface* on suomennettuna käyttöliittymä. Käyttöliittymään liittyy muutakin kuin napit ja kuvakkeet. Käyttäjä tekee olettamuksia sen perusteella, mitä ruudulla esitetään. Suunnittelijan kannattaa pitää mielessä seuraavat adjektiivit: selkeys, ytimekkyys, tuttavallisuus, responsiivisuus, johdonmukaisuus, esteettisyys, tehokkuus ja anteeksiantavuus. Selkeydellä varmistetaan, ettei käyttäjä joudu arpomaan liikaa. Ytimekkyydellä rajoitetaan aivojen prosessointi tarpeita. Tuttavallisuudella hyödynnetään alan tuttuja käytänteitä ja asioiden ulkoasua. Responsiivisuudella pidetään käyttäjä tyytyväisenä, koska hänen ei tarvitse odottaa turhan pitkään, kun hän liikkuu sovelluksessa. Tähän sisältyy myös hyvät käyttäjää informoivat palautteet. Johdonmukaisuudella taataan, ettei käyttäjä hämmenny, jos jokin asia tekeekin yhtäkkiä jotain olettamatonta. Esteettisyyden pitämisellä mielessä, saadaan aikaan näyttäviä tuotoksia. Tehokkuudella pidetään edelleen käyttäjä tyytyväisenä, jotta hän säästää aikaa työskennellessään. Anteeksiantavuudella varmistetaan, että virheiden sattuessa, käyttäjää ei rankaista loukkaavalla tavalla. (User Interface Design in Modern Web Applications 2011.)

Käyttöliittymiä suunniteltaessa tulee miettiä paljon käyttäjän näkökulmaa. Runko ja asettelut ovat vastuussa käyttäjän ohjaamisesta. Yksityiskohtien koolla ja muodolla, voidaan saavuttaa erilaisia tunteita ja merkityksiä. Esimerkiksi painikkeista tulee tehdä tarpeeksi suuria ja niiden pitää myös näyttää painikkeilta, jotta merkitys on lähes itsestään selvä. Väriteoria on ollut kautta aikojen suuressa merkityksessä. Jokainen väri sisältää jonkinlaisen tunteen. Värisokeus tulee myös huomioida, jolloin kontrastin merkitys nousee suureen rooliin. Tekstuureilla voidaan luoda merkityksiä, jotka helpottavat käyttäjää liittämään kontekstin johonkin arkielämän tapaukseen. (User Interface Design in Modern Web Applications 2011.)

Koska värit ovat tärkeä osa tunnelmaa, on hyvä tietää millaisia tunteita eri värit herättävät käyttäjässä. Punainen on lämmön, intohimon ja rakkauden väri. Myös viha

tai aggressiivisuus yhdistetään siihen usein. Oranssi on lämmin ja leikkisä, vähemmän räjähtävä kuin punainen. Keltainen kuvaa onnellisuutta, toivoa ja spontaanisuutta. Vihreän pääpiirteet ovat luonto, kasvu ja harmonia. Sininen luo rauhaa, luotettavuutta ja älykkyyden tunteita. Violetti kattaa luksus, mystisyys ja hengellisyys -tunteet. Vaaleanpunainen on naisellisuuden, leikkisyyden ja romanttisuuden väri. Ruskean tunteet ovat hyveellisyys, lämpöisyys ja rehellisyys. Musta on voiman, tyylikkyyden ja hienostuneisuuden kokonaisuus. Valkoinen taas puhtauden, viattomuuden ja yksinkertaisuuden. Harmaa näiden kahden välimaasto eli ammattimaisuuden, muodollisuuden ja tavanomaisuuden väri. Kulta, hopea ja pronssi ovat varallisuuden, menestyksen ja voitollisuuden värit. (Lundberg 2019.)

3 Suunnittelu

3.1 Aikaisemman toteutuksen ongelmat

Aikaisempi React Native -toteutus Z-Builder -sovelluksesta oli lakannut toimimasta kokonaan. Se oli myös visuaalisesti melko vanhentuneen näköinen, eikä brändi-imago ollut yhdistettynä kokonaisuuteen. Vanhan version ulkoasun rakenteesta sai kuitenkin hyvän pohjan, jonka avulla pystyi nopeammin suunnittelemaan uudistettavaa ulkoasua ja runkoa.

Vanha toteutus toimi MySQL-ympäristön kautta ja siinä oli todettu hankaluuksia saada uusia toimintoja aikaiseksi. MongoDB:tä oli tarpeellista tutkia mahdollisena vaihtoehtona, koska sen rakenteeseen on nopeampi toteuttaa muutoksia.

Vanhassa toteutuksessa ei myöskään ollut mukana Unity-versioita, joita on tärkeä pystyä seuraamaan ja muuttamaan yksittäisen projektin asetuksista.

Unity on suosituin pelimoottori, jossa on paljon omaisuuksia ja laaja latausgalleria, josta voi lisätä toiminnallisuutta sekä pelielementtejä. Unityllä voi tehdä paljon asioita täysin ilmaiseksi. (Petty n.d.)

3.2 MVP eli minimal viable product

Vuoden 2020 tammikuun alkupuolella järjestettiin aloituspalaveri, jossa kasattiin Trelloon uuden toteutuksen *MVP*-vaatimuksia. Vaatimuksiin kuuluivat: toimiva Vue Native frontend, MongoDB-, Node- ja Express-backend, Google-authentikaatio, että vain tietyt Google tilit pääsevät sovellukseen. Lisäksi toimintoina tuli saada toteutettua projektien listaus, projektien lisääminen, iOS ja Android -sovellusversioiden asettaminen rakennettavaksi, Unity-versioiden lisääminen ja poistaminen. Myöhemmin lisättiin projektien poistaminen mukaan tarvittavaksi ominaisuudeksi. Android-versiot vaativat *keystore*-tiedoston, joka sisältää salausavaimia tai muita sertifikaatteja, jotta ne voidaan ladata kauppaan ladattavaksi (Sign your app 2019). Tätä mietittiin sovellukseen mukaan, mutta se osoittautui hankalaksi ja jätettiin toistaiseksi sivuun.

3.3 Nykyaikaistaminen

Sovelluksissa on nykyään paljon mietittävää, jotta se tuntuu laadukkaalta. Ainoastaan toimivuudella ei aina pysty vakuuttamaan käyttäjää. Hyviä käytänteitä seuraamalla voi varmistaa, että käyttäjän kynnys sovelluksen oppimiselle on mahdollisimman matala. Vuoden 2019 lopussa Zaibatsu uusi brändi-imagoaan, joka haluttiin saada näkyviin myös Z-builderissa. Tästä syystä oli helppo valita modernit fontit *Oswald* ja *Open-Sans* käyttöön. Projektin aikana tutkittiin myös muita sovelluksia ja etsittiin niistä samankaltaisuuksia.

Animaatioilla saadaan elävöitettyä sovellusta, joka nostaa sen laadun tuntua uudelle tasolle. Sovellukseen lisättiin projektilistaan pyörivä indikaattori kuvaamaan, että tie-

tystä projektista on käynnissä rakentaminen. Tarkemmalle projektisivulle lisättiin liikkuvuutta nappeihin, jotta käyttäjä saa tunteen, että asioita tapahtuu, eikä hän ole tietämätön etenemisestä. Projektinäyttöön lisättiin myös tilannepalkki, josta voi helposti seurata valmistumista. Tilanneseuranta on myös nähtävissä etusivun listassa prosenttiosuuden muodossa. Autentikaation ja projektilistan päivittämisen yhteyteen lisättiin indikaattori, joka pyörii silloin, kun asioita suoritetaan taustalla.

Väreinä käytettiin uusia brändivärejä, jotka on esitetty kuviossa 15.

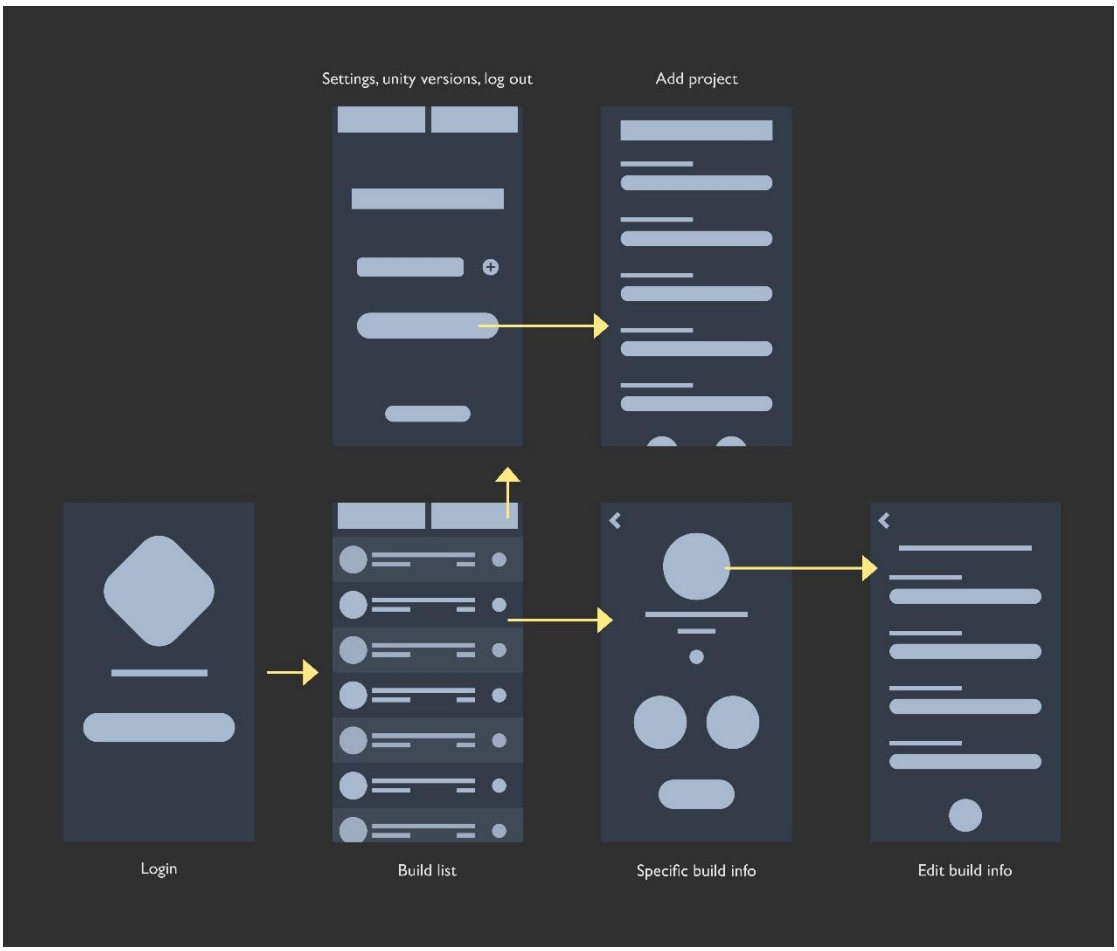
Kuvio 15 Zaibatsu Interactive Oy brändivärit

3.4 Näyttöhierarkia

Näyttöjen hierarkia on tärkeä miettiä kunnolla, jotta käyttäjä ei eksy käyttäessään sovellusta. Alapuolen kuviossa 16 on lähtötilanne ruutujen yhteyksistä. Projektin aikana tuli tarpeita poiketa alkuperäisestä suunnitelmasta osittain.

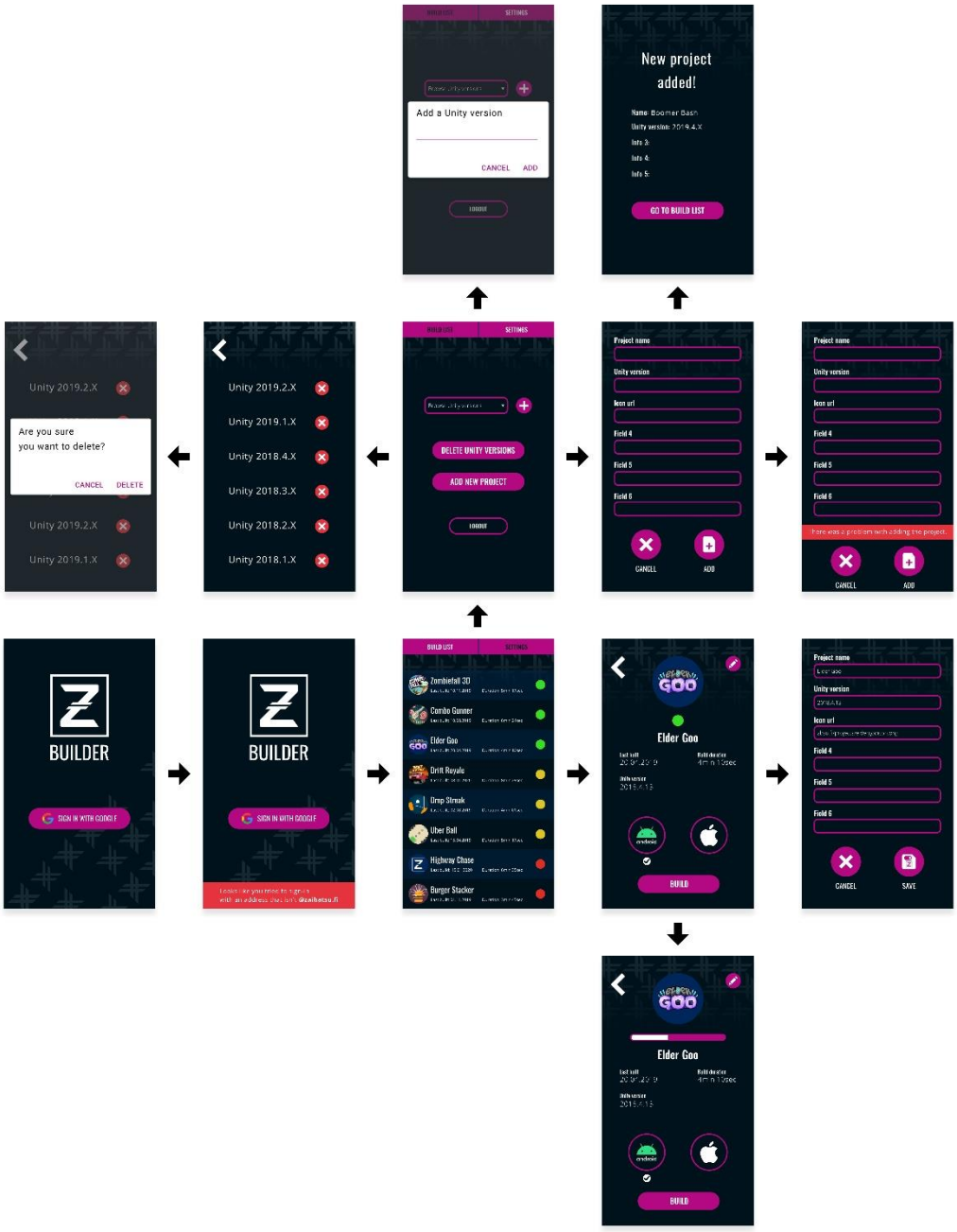
Kuvio 16 Z-builder näyttöjen etenemisjärjestys

Kuviossa 17 on esitetty alussa tehtyä suunnitelmaa, jossa haettiin muotoja ja asettelua sovellukselle. Sovelluksessa oli tärkeä varmistaa, että perusajatus on kunnossa ennen kuin lähdettiin keskittymään yksityiskohtiin liikaa. Myös liikkumista alettiin miettiä jo alkuvaiheessa.



Kuvio 17 Z-builder suunnanhaku suunnitelma

Kuviossa 18 on lopullinen hyväksytty toteutus, josta selviävät kaikki tarvittavat elementit ja näytöt. Muutoksia tehtiin, kun ajatus selveni entisestään. Kokonaisuudessaan suunnitelma oli onnistunut ja sen pohjalta oli helppo lähteä teknisen toteutuksen pariin.



Kuvio 18 Yksityiskohtainen Z-builder UI-suunnitelma

3.5 Käytettävyys

Käytettävyys on osoittautunut tärkeäksi nykyajan osa-alueeksi viime vuosina ja se haarautuu tiuhaan uusiin rooleihin ja tehtäviin. Z-Builderia kehittäessä pidettiin alati mielessä, millaisia tuntemuksia käyttäjä kokee eri tilanteissa. Tämä auttoi luomaan hyviä ja toimivia ratkaisuita.

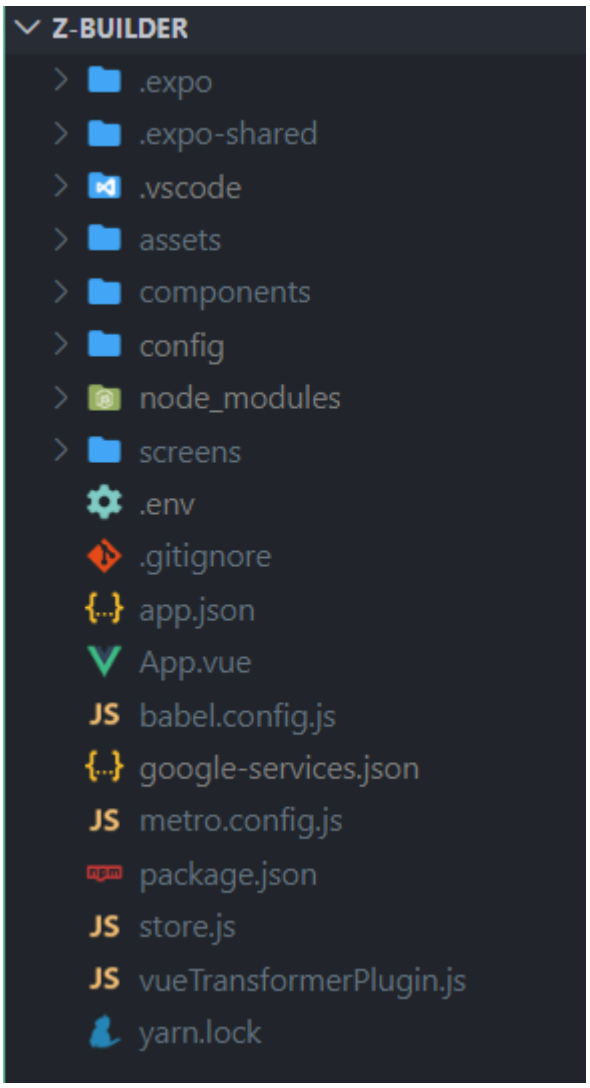
Tärkeimpinä olivat informaation ryhmittely ja yleinen ymmärrettävyys. Painikkeiden tulee olla selkeitä ja havainnollisia. Niistä tehtiin myös tarpeeksi isoja, koska puhelimella on vaikea osua pieniin elementteihin. Käyttäjää tulee myös informoida tarpeeksi usein erilaisilla ilmoituksilla tai muuttuvilla yksityiskohdilla. Teksteissä keskityttiin luettavuuteen ja yksinkertaisuuteen. Informaatiopaljoutta pyrittiin välttämään, missä se oli mahdollista.

4 Tekninen toteutus

4.1 Vue Native alustus ja Expo

Vue Native -projektin aloitus vaatii Nodejs-versiosta ainakin version kuusi ja npm:stä version neljä tai ylemmän. Expo CLI tai React Native CLI tulee myös olla asennettuna. Kaikki nämä tulee asentaa systeeminlaajuisesti optimaalisen toimivuuden takaamiseksi. *Vue-native-cli* -paketti pitää näiden jälkeen asentaa myös systeeminlaajuisesti, jotta projekti voidaan alustaa. Expo-sovellukset saa helposti testikäyttöön omaan puhelimeen lataamalla Expo-sovelluksen oman puhelimen sovelluskaupasta. Kun terminaalista ajetaan *expo start*, samalla luodaan QR-koodi, jonka kautta sovellus saadaan latautumaan oman puhelimen ruudulle. Tämä nopeuttaa testaamista ja mahdollistaa oikean ympäristön kokemisen ilman suuria ongelmia. Puhelimen tulee kuitenkin olla samassa lähiverkossa, jotta tämä toiminto saadaan käyttöön.

Kuviossa 19 on nähtävissä Z-Builderin kansiorakenne. *Assets*-kansio sisältää fontit ja kuvat, *components*-kansio komponentteja ja *screens* sisältää kaikki näytöt. Koska tekijä oli kokematon, kansiorakenne ei ole optimaalinen ja kaikista selkein mahdollinen. Komponentit-kansiossa on navigaatioon liittyviä asioita, eikä näyttöjä ole jaettu helposti uudelleenkäytettäviin palasiin. Projektin aikana ei kuitenkaan ilmennyt useaa tilannetta, jossa tästä olisi ollut suurta hyötyä. Tulevaisuudessa olisi viisaampi rajata projektia pieniin palasiin, jotta ymmärrettävyys ja uudelleenkäytettävyys paranevat. Tämä on tärkeää varsinkin, jos projektiin tulee uusia kehittäjiä mukaan.



Kuvio 19 Z-builder kansiorakenne

4.2 Näyttökomponentti

Näyttökomponentteja luodaan hyödyntämällä *view*, *touchable-opacity*, *text*, *image*, *button* -elementtejä. *View* vastaa vahvasti web-kehityksestä tuttua div-elementtiä. Se on tärkeimpiä rakennuspilareita Vue Native -kehityksessä.

Kuviossa 20 on pala Z-Builderin projektilistan koodista. Koodissa on tuttuja rakenteita HTML-puolelta ja *class*-nimet ovat mukana tuttuun tyyliin. Vue tuo mukaan *v-if* -ominaisuuden, johon voidaan asettaa ehtoja sille, näytetäänkö osa käyttäjän ruudulla vai ei. Kuvion tilanteessa tarkastetaan, onko Vuex-tilassa *projects*-objekti luotuna, jotta näytölle voidaan tulostaa tietoja. Jos tätä ei varmisteta, sovellus menee usein virhetilaan, eikä käyttäminen ole mahdollista. Kuviosta huomataan myös, että tulosteita voidaan muokata JavaScript keinoin suoraan template-koodissa. Tässä tapauksessa on hyödynnetty *substr*-funktiota, jonka avulla objektissa sijaitsevasta tekstipätkästä saadaan vain haluttu pala käyttöön.

```
<view class="project_info">
    <image
        v-if="project && project.imageUrl && project.imageUrl.includes('https')"
        :source="{uri: project.imageUrl }"
        class="image"
        :style="{width: 50, height: 50}"
    />
    <view v-if="project.imageUrl && !project.imageUrl.includes('https')" class="placeholder_img_style">
        <text class="placeholder_img_text">{{ project.name.slice(0,1) }}</text>
    </view>
        <view class="project_details">
            <view class="project_header_row">
                <text class="project_name_text">{{ project.name }}</text>
                <text v-if="project.status == 'building'" class="project_build_percent">
                    {{ calculateProjectPercent(project.builds.startTime, project.builds.endTime) }}%
                </text>
            </view>
                <view class="side_by_side_details">
                    <text v-if="project.lastBuilt !== 'Not built'" class="project_last_built">
                        {{ project && project.lastBuilt.substr(0,6) }}
                    </text>
                    <text v-if="project.lastBuilt !== 'Not built'" class="project_time">
                        {{ project && project.lastBuilt.substr(project.lastBuilt.length - 5) }}
                    </text>
                    <text v-if="project.lastBuilt !== 'Not built'" class="project_build_duration_text">
                        {{ project.duration }} sec
                    </text>
                    <text v-if="project.lastBuilt === 'Not built'" class="project_build_duration_text">
                        Created {{ project.createdAt.slice(8,10) }}.{{ project.createdAt.slice(5,7) + '.' }}
                    </text>
                </view>
            </view>
```

Kuvio 20 Koodipätkä z-builder projektilistasta

4.3 Navigaatio

Navigaatio on tärkeä osa sovellusta. Vue Native hyödyntää *React Navigation* -kirjastoa sen luomisessa. Kuviossa 21 on kuvakaappaus navigaation alustamisesta. Eri näytöt otetaan käyttöön niiden tiedostopoluista *import*-termiä käyttäen.

```
<template>
    <AppNavigation />
</template>

<script>

import {
  createAppContainer,
  createMaterialTopTabNavigator,
  createStackNavigator,
} from "vue-native-router";

import { Animated, Easing } from "react-native";

import ProjectList from "../screens/ProjectList";
import Settings from "../screens/Settings";
import SpecificProject from "../screens/SpecificProject";
import EditProjectInfo from "../screens/EditProjectInfo";
import AddUnityVersions from "../screens/AddUnityVersions";
import RemoveUnityVersions from "../screens/RemoveUnityVersions";
import AddNewProject from "../screens/AddNewProject";
import NewProjectAdded from "../screens/NewProjectAdded";
import SignInScreen from "../screens/SignInScreen";
import LoadingScreen from "../screens/LoadingScreen";
```

Kuvio 21 Navigaation alustus

Kuviossa 22 nähdään, miten eri näytöistä koostetaan *StackNavigator*. StackNavigator toiminta mahdollistaa sovelluksessa edestakaisen liikkumisen. Aina kun sovelluksessa liikutaan takaisin, StackNavigatorin ylin näyttö siirretään pois jonosta.

```
69  const SettingsStack = createStackNavigator({
70      Settings: {
71        screen: Settings
72      },
73      AddUnity: {
74        screen: AddUnityVersions
75      },
76      RemoveUnity: {
77        screen: RemoveUnityVersions
78      },
79      AddProject: {
80        screen: AddNewProject
81      },
82      NewProjectAdded: {
83        screen: NewProjectAdded
84      }
85  },
```

Kuvio 22 StackNavigator asetuksia

Kuviossa 23 on *TopNavigation*-konfiguraatio. Näyttöjä voidaan asettaa samalla tavalla kuin StackNavigatorissa, mutta TopNavigaation kaikki ruudut tulevat toimiviksi vierekkäin. Jokaiselle ruudulle ei kuitenkaan tarvitse asettaa otsikkoa, jos ei halua, että se esiintyy yläpalkissa. TopNavigation mahdollistaa sulavan liikkumisen kahden tai useamman sivun välillä, esimerkiksi vetämällä sormella oikealle tai vasemmalle. Ruudun ylälaitaan luodaan myös otsikot, joita painamalla voidaan siirtyä haluttuun ruutuun.

```
const TopNavigation = createMaterialTopTabNavigator(
  {
    Project: { screen: ProjectStack, navigationOptions: { tabBarLabel: "Projects List" } },
    Settings: SettingsStack
  },
  {
    initialRouteName: "Project",

    tabBarOptions: {
      activeTintColor: 'white',
      inactiveTintColor: '#00141E',
      style: {
        backgroundColor: '#B40A82',
        paddingTop: 32,
      },
      labelStyle: {
        fontFamily: "Oswald-Regular",
        fontSize: 20,
      },
      indicatorStyle: {
        backgroundColor: 'white',
      },
    }
  }
);

const AppNavigation = createAppContainer(TopNavigation);

export default {

  components: { AppNavigation }

}
```

Kuvio 23 TopNavigation asetuksia

Jotta navigoinnin yhteydessä voidaan kuljettaa tietoa, Vue:ssa pitää alustaa *propsit* komponenttiin. Sen avulla voidaan kuljettaa esimerkiksi ID-tieto halutulle alaruudulle. Kuviossa 24 on tehty juuri tämä operaatio, kun liikutaan tietyn projekti yksityiskohtaiseen näyttöön. Jos tietoja pitää siirtää monta näyttöä syvälle, tavasta tulee työläs. Tämän seurauksena voi miettiä, olisiko projektissa tarvetta ottaa tilanhallinta käyttöön ja keskittää tietoja yhteen paikkaan.

```
120     props: {
121         navigation: {
122             type: Object
123         }
124     },
125
126     methods: {
127         goToProject(item) {
128             this.navigation.navigate('SpecificProject', {projectID: item._id})
129         },
```

Kuvio 24 Navigaatio propsit

4.4 Vuex Z-Builderissa

Z-Builderissa käytettiin Vuex-tilanhallintajärjestelmää apuna. Sen avulla suurin osa tiedoista saatiin pidettyä yhdessä keskitetyssä paikassa. Vuex alustetaan yleensä *store.js*-tiedostoon projektin juureen. Joskus voi olla hyödyllistä jakaa erinäiset toiminnallisuudet myös omiin kansioihin, mutta tässä projektissa tälle toimenpiteelle ei ollut tarvetta. Mitä enemmän Vuex:ltä vaaditaan, sitä pidemmäksi tiedosto laajenee ja luettavuus kärsii. Siksi voi olla parempi erotella osia eri tiedostoihin.

Kuviossa 25 on esitetty varaston ydin, jossa on lueteltu säilömistä vaativat osat. Tärkeimpiä näistä ovat *projects*, johon ladataan tietokannasta kaikki projektit *ja currentProject*, jonka avulla valittua projektia päivitetään näytölle.

```
import Vue from 'vue-native-core';
import Vuex from 'vuex';
import { localip } from './config/ipconfig.json'

Vue.use(Vuex);

export const store = new Vuex.Store({
    state: {
        projects: [],
        currentProject: [],
        currentImage: '',
        projectsStatus: '',
        infoMessage: '',
        loadingImages: null,
        userHelpPrompt: ''
    },
```

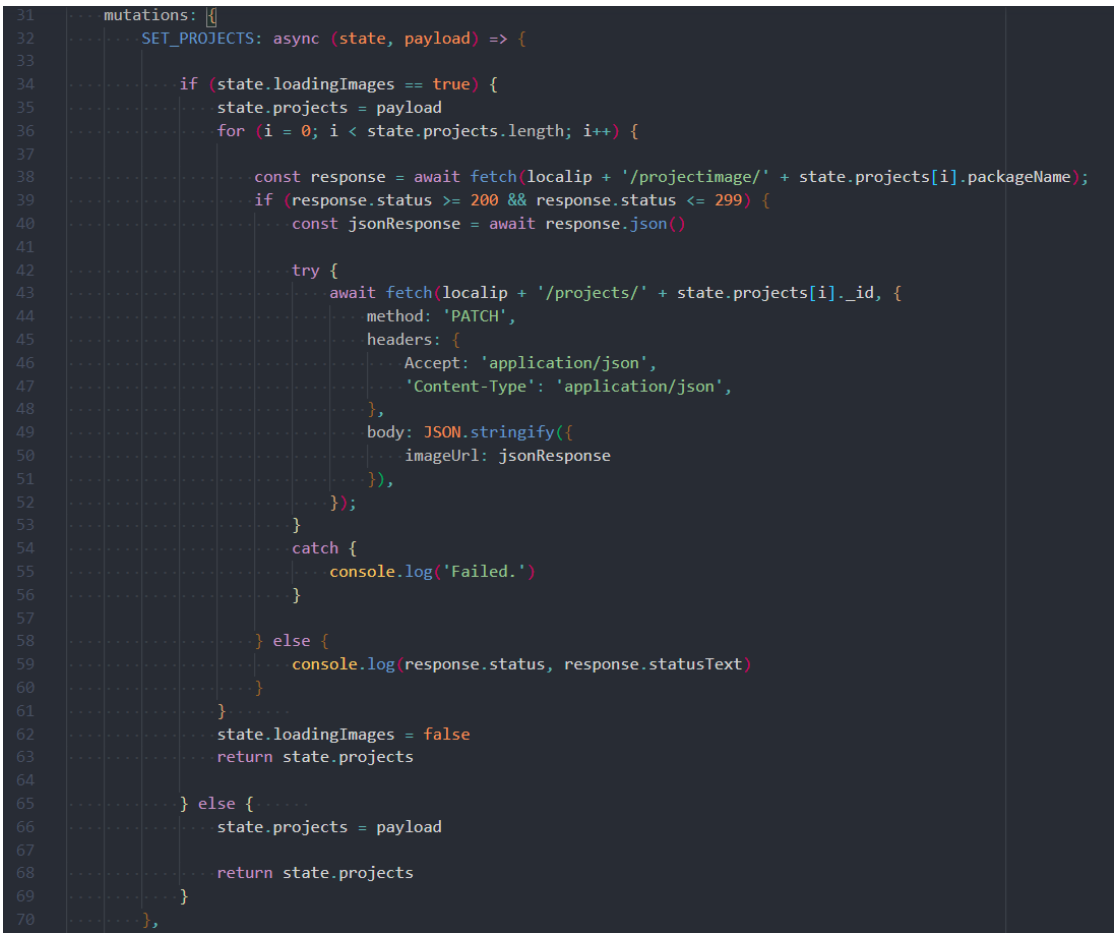
Kuvio 25 Vuex store

Vuex:ssä *actions*-osan kautta suoritetaan *mutations*-toimintoja, jotta *state*-tilaa ei muunneta suoraan. Tämä helpottaa virheiden etsimistä debug-toimintojen kautta. Z-Builderin tapauksessa kohta *GET_PROJECTS* hakee API:n kautta kaikki projektit, ja muokkaa tilaa saadun tiedon perusteella. Kuviossa 26 API:n kautta haetaan projektit tietokannasta ja annetaan mutaatiolle tehtäväksi lisätä tiedot Vuex-tilaan.

```
104     actions: {
105         GET_PROJECTS: async (context, payload) => {
106             const response = await fetch(localip + '/projects/');
107             if (response.status >= 200 && response.status <= 299) {
108                 const jsonResponse = await response.json();
109                 context.commit('SET_PROJECTS', jsonResponse)
110             } else {
111                 console.log(response.status, response.statusText);
112             }
113         },
114     }
```

Kuvio 26 Vuex store action

Kuviossa 27 haetaan tiedot uusiksi, mikäli projektien kuvat pitää ladata uudestaan. Kuvat haetaan Google Play -sivulta käyttämällä projektin *package-name* tietoa. Tätä toimintoa varteen projektiin luotiin *scraper*, joka leikkaa verkkosivulta vain halutut osat. Tässä tapauksessa projektin kuvaosoitteen. Projektien tiedonhaku-funktio pitäisi hyvän käytänteen mukaan olla *actions*-osassa, mutta se huomattiin vasta myöhemmin, eikä voi olla varma miten helppo muutos on kyseessä.



```
mutations: {
    SET_PROJECTS: async (state, payload) => {

        if (state.loadingImages == true) {
            state.projects = payload
            for (i = 0; i < state.projects.length; i++) {

                const response = await fetch(localip + '/projectimage/' + state.projects[i].packageName);
                if (response.status >= 200 && response.status <= 299) {
                    const jsonResponse = await response.json()

                    try {
                        await fetch(localip + '/projects/' + state.projects[i]._id, {
                            method: 'PATCH',
                            headers: {
                                Accept: 'application/json',
                                'Content-Type': 'application/json',
                            },
                            body: JSON.stringify({
                                imageUrl: jsonResponse
                            }),
                        });
                    }
                    catch {
                        console.log('Failed.')
                    }

                } else {
                    console.log(response.status, response.statusText)
                }
            }
            state.loadingImages = false
            return state.projects

        } else {
            state.projects = payload

            return state.projects
        }
    },
```

Kuvio 27 Vuex store mutation

Kuviossa 28 on esitelty erilaisia *getters*-ominaisuuksia, joiden avulla voidaan hakea tietoa Vuex-statesta. Muitakin tapoja on, mutta selkeät rajaukset helpottavat, jos on monta kehittäjää samassa projektissa. Getters ja setters -ominaisuudet ovat tuttuja käytänteitä olio-ohjelmoinnissa.

```
17    getters: {
18        PROJECTS : state => {
19            return state.projects
20        },
21        CURRENT: state => {
22            return state.currentProject
23        },
24        CURRENT_BUILD_NUMBER: state => {
25            return state.currentProject.builds.buildNumber
26        },
27        POPUP_MESSAGE: state => {
28            return state.userHelpPrompt
29        }
30    },
```

Kuvio 28 Vuex store getters

4.5 Animaatioiden luonti

Animaatioiden luominen on Vue Nativessa helppoa käyttäen React Nativen *Animated* ja *Easing* osia. Ajatus voi aluksi olla hieman omituinen, mutta kun sen oppii, animaatioita saa luotua nopeasti haluttuihin käyttötarkoituksiin. Kuviossa 29 on esimerkki siitä, miten animaatiot alustetaan, kun komponentti luodaan.

```
489    created() {
490        this.progressOpacityValue = new Animated.Value(0)
491        this.opacityValue = new Animated.Value(1)
492        this.moveValue = new Animated.Value(0)
493        this.spinValue = new Animated.Value(0)
494        this.wiggleValue = new Animated.Value(0)
495    },
```

Kuvio 29 Animaation alustus

Animaatioita voidaan ketjuttaa, jolloin elementti saadaan liikkumaan ensin oikealla ja sitten vasemmalle. Kuviossa 30 on luoto tällainen ketjutus. *Easing* huolehtii vähemmän lineaarisesta siirtymisestä. Easing-animaatiot näyttävät mielenkiintoisimmilta ja ovat persoonallisia verrattuna lineaarisiin. Oudompana ominaisuutena kuviossa esiintyy myös *useNativeDrive*, jonka avulla voidaan kertoa puhelimelle, että sen tulee käyttää omaa ohjaintaan, jotta animaatio toimii tehokkaasti. Kaikille animaatioille tätä ei voida asettaa. Jokainen animaatio tulee myös aloittaa kutsumalla lopussa *start()*-funktiota.

```
310     wiggleAnimation() {
311         this.wiggleValue.setValue(0);
312
313         Animated.sequence([
314             Animated.timing(
315                 this.wiggleValue,
316                 {
317                     toValue: -4,
318                     duration: 200,
319                     easing: Easing.ease,
320                     useNativeDriver: true
321                 }),
322                 Animated.timing(
323                     this.wiggleValue,
324                 {
325                     toValue: 0,
326                     duration: 200,
327                     easing: Easing.out(Easing.ease),
328                     useNativeDriver: true
329                 }),
330         ])
331         .start();
332     },
```

Kuvio 30 Animaation asetukset

Kuviossa 31 on esitetty, miten animaatiota kutsutaan funktioiden sisällä.

```
185        toggleAndroid() {
186            this.androidChecked = !this.androidChecked
187            this.wiggleAnimation()
188        },
189
190        toggleIOS() {
191            this.iosChecked = !this.iosChecked
192            this.wiggleAnimation()
193        },
```

Kuvio 31 Animaation ajaminen

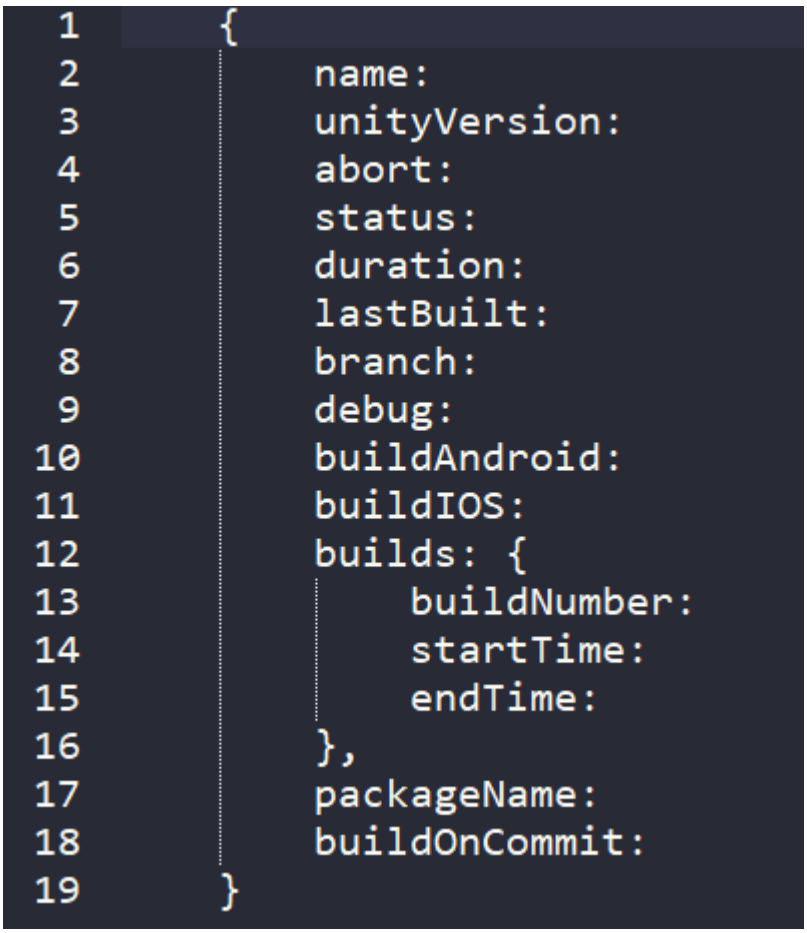
4.6 Node, MongoDB ja Express -yhteys

Node-palvelin on äärimmäisen yksinkertainen alustaa. Kuviossa 32 otetaan ensin tarvittavat paketit käyttöön. Sen jälkeen palvelin yhdistetään Mongoon. Tämän jälkeen app-muuttujan kautta ajetaan Express, joka luo API-polut. Lopuksi palvelin laitetaan toimimaan porttiin 3000.

```
const express = require('express')
const app = express();
const mongoose = require('mongoose');

mongoose.set('useCreateIndex', true);

const mongoDB = 'mongodb://127.0.0.1/testingdb';
mongoose.connect(mongoDB, { useNewUrlParser: true, useUnifiedTopology: true });

const db = mongoose.connection;

db.on('error', console.error.bind(console, 'MongoDB connection error:'));

db.once('open', () => console.log('Connected to mongo database!'))

app.use(express.json())

const projectsRouter = require('./routes/projects')
app.use('/projects', projectsRouter)

const unityRouter = require('./routes/unityversions')
app.use('/unityversions', unityRouter)

const imageRouter = require('./routes/imageurlparse')
app.use('/projectimage', imageRouter)

app.listen(3000, () => console.log('Server started on port 3000.'))
```

Kuvio 32 MongoDB, Node ja Express yhteys

MongoDB alustettiin kuvion 33 mukaisilla tietueilla eri testiprojekteille. Unity-versioille luotiin samankaltainen dokumentti, jossa ainoana sarakkeena oli versiotieto.



Kuvio 33 MongoDB datamalli

Jokainen tietokanta kokoelma pitää kartoittaa Noden päässä oikein. Mongoose hoitaa tämän, kunhan kokoelmalle on luotu *schema*. Tämän avulla palvelin osaa käsitellä dataa oikealla tavalla, ja tietää mitä arvoja tietueet saavat sisältää. Kuviossa 34 on esimerkki Unity-skeemasta. Skeemaan tulee myös kertoa, mihin MongoDB-varastoon tietoa tullaan säilömään.

```
const mongoose = require('mongoose')

const unitySchema = new mongoose.Schema({
    version: {
        type: String,
        unique: true,
    }
},
{
    collection: 'unitycollection',
    timestamps: { createdAt: 'createdAt' }
});

module.exports = mongoose.model('Unity', unitySchema)
```

Kuvio 34 Mongoose malli Unity-versiosta

4.7 API

API-toimintoja luotiin projektin hakemiselle, poistamiselle ja muokkaukselle. Tässä apuina ovat *get, post* ja *patch*. Unity-version lisäämiselle ja versioiden poistamiselle luotiin myös vastaavat toiminnot. Projektin ID-tiedon avulla voidaan päivittää oikeat kohdat uusille arvoille. Kuviossa 35 on koodiote, jossa tarkistetaan omaako saapunut palaute projektinimeä. Jos omaa, projekti päivitetään tällä uudella nimellä.

```
45  router.patch('/:id', getOneProject, async (req, res) => {
46
47      if (req.body.name != null) {
48          res.project.name = req.body.name
49      }
```

Kuvio 35 Tietojen päivittämisen polku

Kuvion 36 koodi huolehtii tiedon päivittämisestä lopullisesti. Jos päivittäminen ei onnistu, annetaan tieto tapahtuneesta virheestä.

```
93      try {
94          const updatedProject = await res.project.save()
95          res.json(updatedProject)
96      } catch(err) {
97          return res.status(400).json({ message: err.message })
98      }
```

Kuvio 36 Tietojen tallentaja Express-koodi

Kuvion 37 koodi on välikäsi, joka etsii oikean projekti käyttäen projektin ID-tietoa. Tämä pala vähentää tarvittavan koodin kirjoitusmäärää ja tekee koodista luettavampaa.

```
async function getOneProject(req, res, next) {
  try {
    project = await Project.findById(req.params.id)
    if (project == null) {
      return res.status(404).json({ message: 'Cant find that project.'})
    }
  } catch(err){
    return res.status(500).json({ message: err.message })
  }

  res.project = project
  next()
}
```

Kuvio 37 Yhden projektin noutamisen apufunktio

4.8 Google kirjautuminen

Kun sovellus käynnistetään ensimmäisen kerran, käyttäjä viedään autentikaatio-näytön kautta itse sovellukseen. Käyttäjä kirjautuu käyttämällä Google-tiliä. Sovellukseen on myös asetettu tarkistus, joka katsoo, onko kyseinen Google-tili zaibatsu.fi-osoitetta käyttävä. Jos on, käyttäjä pääsee sovellukseen. Kuviossa 38 on sovelluksen autentikaatio-näkymä.

Kuvio 38 Autentikaatio ruutukaappaus

Kuviossa 39 käydään läpi Firebase-lista ja tarkastetaan, onko käyttäjä jo lisätty. Kuviossa 40 käyttäjä lisätään Firebase-palvelun listaan, mikäli käyttäjää ei siellä vielä ole olemassa. Kuviossa 41 käyttäjä päästetään sovellukseen, mikäli aiemmat tarkastukset läpäistiin. Nämä koodiohjeet löytyvät Firebase-dokumentaatiosta, mutta niitä on muutettava hieman, jotta ne toimivat Vue Nativessa. Ainakin *this* tulee sitoa funktiolle erikseen, jotta se viittaa oikeaan asiaan. Vastaavasti funktioita voi muuttaa käyttämään nuolisyntaksia.

```
39        isUserEqual(googleUser, firebaseUser) {
40            if (firebaseUser) {
41                var providerData = firebaseUser.providerData;
42                for (var i = 0; i < providerData.length; i++) {
43                if (providerData[i].providerId === firebase.auth.GoogleAuthProvider.PROVIDER_ID &&
44                    providerData[i].uid === googleUser.getBasicProfile().getId()) {
45                    return true;
46                }
47                }
48            }
49            return false;
50        },
```

Kuvio 39 Firebase käyttäjän tarkistus

```
52        onSignIn(googleUser) {
53            console.log('Google Auth Response', googleUser);
54            var unsubscribe = firebase.auth().onAuthStateChanged(function(firebaseUser) {
55                unsubscribe();
56                if (!this.isUserEqual(googleUser, firebaseUser)) {
57                    var credential = firebase.auth.GoogleAuthProvider.credential(
58                        googleUser.idToken,
59                        googleUser.accessToken
60                    );
61                    firebase.auth().signInWithCredential(credential).catch(function(error) {
62                        var errorCode = error.code;
63                        var errorMessage = error.message;
64                        var email = error.email;
65                        var credential = error.credential;
66                    });
67                } else {
68                    console.log('User already signed-in Firebase.');
69                }
70            }.bind(this));
71        },
```

Kuvio 40 Toimenpiteet uudelle käyttäjälle

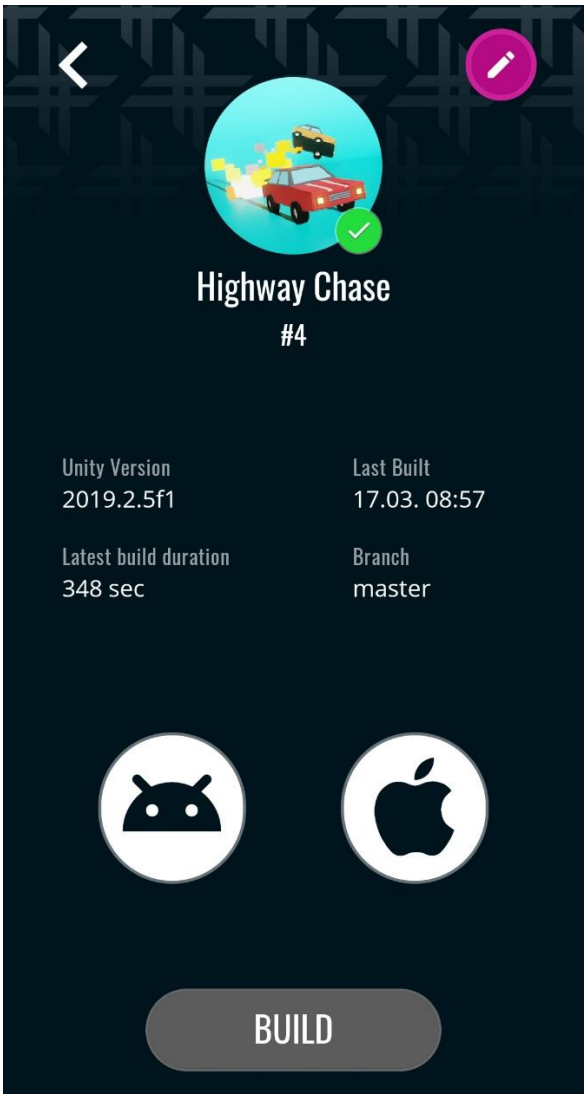
```
async signInToGoogle() {
    try {
        const result = await Google.logInAsync({
        androidClientId: ANDROID_CLIENT_ID_KEY,
        scopes: ['profile', 'email'],
        });

        if (result.type === 'success') {
            this.onSignIn(result)
            this.navigation.navigate('Loading')
            return result.accessToken;
        } else {
            return { cancelled: true };
        }
    } catch (e) {
        return { error: true };
    }
},
```

Kuvio 41 Lopullinen sisäänkirjaus

4.9 Sovellusten kokoamispyynnöt Z-Builderilla

Niin sanottu *buildaaminen,* eli sovelluksen kokoaminen koodista puhelimessa toimivaksi kokonaisuudeksi, tapahtuu Z-Builderissa asettamalla tietokantaan tieto siitä, onko tilattu Android tai iOS -versioiden rakennusoperaatio. Tietokanta tulee tulevaisuudessa olemaan yhteydessä Jenkins-palveluun, joka seuraa tietokantaan tapahtuvia muutoksia ja alkaa sen perusteella suorittamaan toimenpiteitä. Tuloksena saadaan toimiva sovellustiedosto, jonka voi asentaa puhelimeen. Sovellustiedosto ei tule Z-Builderiin, vaan se lähetetään saataville käyttäen muuta reittiä. Usein valmiit sovellustiedostot ladataan pilvipalveluun helpottamaan jakelua. Z-Builder -mobiilisovellus toimii vain pyyntöjen välikätenä.

Kuviossa 42 näkyy projektin yksilöllinen näkymä, josta voidaan lähettää tietokantaan tieto, mille alustalle sovellustiedostoja pitää rakentaa. Projektin nimen alapuolelle tuodaan tieto, monesko rakentamiskerta on menossa. Kuvion tilanteessa projekti on rakennettu 4 kertaa. Alustakuvakkeista, iOS tai Android, painamalla voidaan valita, kumpi halutaan. Kun painikkeista painetaan, alapalkki vaihtuu ilmoittamaan, että pyyntö voidaan laittaa eteenpäin. Kun pyyntö on lähetetty, alustalogot häivytetään ja ruudulle tuodaan palkki, josta voidaan seurata arvioitua valmistumisaikaa. Rakentamisen ajaksi näyttöön tuodaan myös painike, jolla prosessi voidaan lopettaa kesken. Tietokannassa on kenttä nimeltä *abort,* joka voidaan vaihtaa todeksi.



Kuvio 42 Projektin buildausnäkymä

Kuviossa 43 on esitetty, miten oikean projektin kohdalle saadaan tietokantaan haluttu pyyntö. Sovelluksesta lähetetään API-polkuun *patch*-muotoinen kysely, jossa tarjotaan uusia tietoja *body*-osiossa. Yhden rakentamisen kesto on testitilanteessa 15 sekuntia aloitusajankohdasta.

```
418             await fetch(localip + '/projects/' + this.projectID, {
419                 method: 'PATCH',
420                 headers: {
421                     Accept: 'application/json',
422                     'Content-Type': 'application/json',
423                 },
424                 body: JSON.stringify({
425                     builds: {
426                         startTime: Date.now(),
427                         endTime: Date.now() + 15000,
428                         buildNumber: this.buildNumber
429                     },
430                     buildIOS: this.iosChecked,
431                     buildAndroid: this.androidChecked,
432                     status: 'building'
433 
434                 })
435             });
```

Kuvio 43 Sovelluskokoamisen fetch-kysely pyyntö

4.10 Projektin lisääminen

Projektin lisääminen sovellukseen tapahtuu lomakkeen kautta. Vue Nativen *TextInput* -kentistä saadaan *v-model* -attribuuttia käyttämällä tieto käyttäjän tekemistä muutoksista, tässä tapauksessa kirjoittamisesta. Kun käyttäjä painaa lisää-nappia, tieto kulkee API:sta tietokantaan ja tuloksena uusi projekti on lisätty kantaan. Kuvissa 44 suoritetaan *post*-pyyntö käyttämällä komponentin *state*-tietoja. Nämä tiedot päivittyvät, kun kirjoitetaan tekstikenttiin.

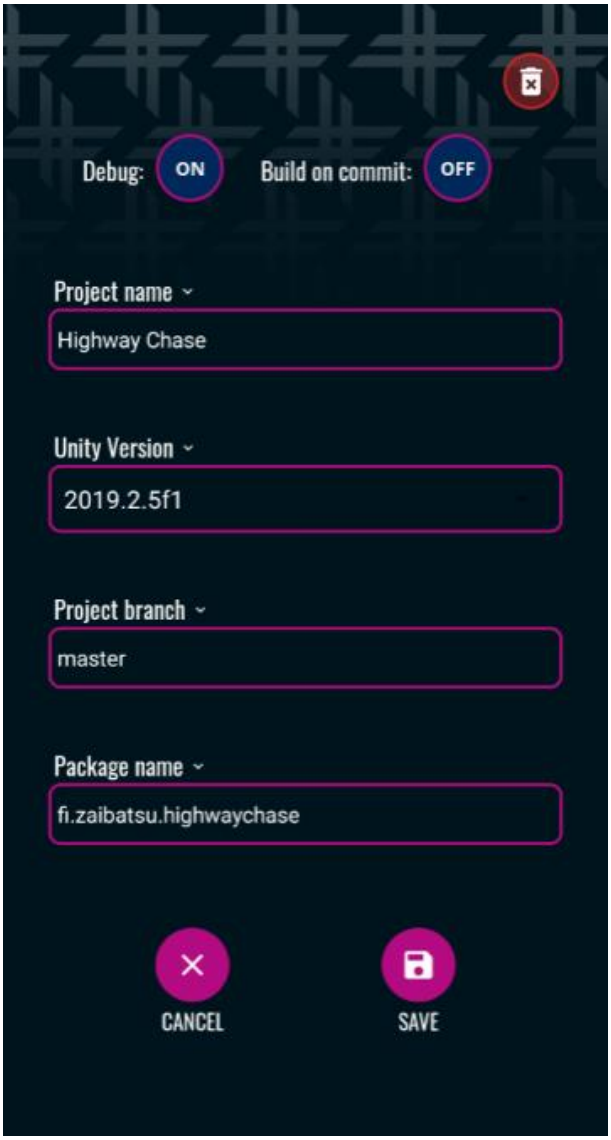
```
154                fetch(localip + '/projects/', {
155                    method: 'POST',
156                    headers: {
157                        Accept: 'application/json',
158                        'Content-Type': 'application/json',
159                    },
160                    body: JSON.stringify({
161                        name: this.projectName,
162                        unityVersion: this.selectedUnity,
163                        branch: this.branchName,
164                        packageName: this.packageName,
165                        debug: this.debugMode,
166                        buildOnCommit: this.buildOnCommit
167                    }),
168                });
169            }
```

Kuvio 44 Projektin lisäämisen post-pyyntö

4.11 Projektin muokkaaminen

Projektien tietoja voidaan muokata siirtymällä projektin ruutuun ja painamalla sieltä muokkauskuvaketta. Tämän jälkeen sovellus siirtyy uuteen näyttöön, jossa *TextInput* -kenttiin voidaan kirjoittaa muutokset ja tallentaa tiedot tietokantaan. Projektin

muokkaaminen toimii samalla tapaa *patch*-pyynnön kautta kuin rakentamispyyntöjen aloittaminen. Kuviossa 45 on kuvakaappaus sovelluksen muokkausnäkymästä.



Kuvio 45 Projektin muokkausnäkymä

5 Tulokset ja pohdinta

Opinnäytetyön tuloksena Vue Nativen toimintaa saatiin kartoitettua oikeassa tilanteessa sovelluskehityksen kautta. Sovellus on moderni ja se tekee asiat, joita sille on rajattu. Vue Native todettiin helpommin lähestyttäväksi kuin React Native, mutta vaativammat käyttötapaukset vaativat React Nativen osaamista taustalle. Vue Nativen kanssa on helppo lähteä kokeilemaan, vaikka taustaa mobiilikehityksestä ei olisikaan. HTML-läheinen lähestymistapa on myös enemmän suunnitteluun painottuville tekijöille selkeämpi. Rakennetussa sovelluksessa mukana oli myös Expo, joka helpottaa tekemistä ja ajattelua vielä enemmän. Expo todettiin hyväksi, mutta ajoittain rajoittavaksi työkaluksi. Erityisesti sovelluksen ajaminen Expon rakennustyökalun läpi vaati tuntien odotusta, ennen kuin sovelluksen sai ladattua ja asennettua puhelimeen.

Suurimmaksi hankaluudeksi työssä nousi koodin suunnittelu. Vilkkaan kehityksen tahdissa ei aina ollut mahdollisuutta miettiä asioita tarkkaan. Aikataulu tuntui tiukalta, koska pienessä ajassa tuli oppia paljon ja saada asioita valmiiksi. Monet näyttökomponenteista rönsyilevät ja sisennykset tekevät koodista vaikealukuista. Selkeyteen sekä jaotteluun olisi pitänyt pyrkiä jo alkumetreillä. Vue Native -projektin aloituksessa kannattaa miettiä tarkkaan, mitkä osat ovat uudelleenkäytettäviä.

Jatkokehityksessä tulisi keskittyä koodin parempaa jaotteluun ja Vue:n vahvuuksien parempaan hyödyntämiseen komponenttien tarkan uudelleenkäytön kautta. Siirtymistä MongoDB -backendiin ei projektin aikana saatu suoritettua, joten Z-Builder jäi testidata vaiheeseen. Tulevaisuudessa on kuitenkin helppo saada sovellus käyttöön, kun tarvittavat backend-muutokset on suoritettu.

Opinnäytetyössä onnistuttiin informaation keruussa ja toimeksiantajalle jäi hyvä kuva, mihin Vue Native kykeni. Nopea oppiminen ja ratkaisuiden ripeä aikaansaaminen olivat toimeksiantajalle arvostettuja huomioita. Myös tekijän oma kyky kysyä apua oli tärkeä etenemisen kannalta. Muuten tekeminen olisi voinut hidastua pitkäksi aikaa. Toimeksiantajan kanssa yhteistyö pelasi ja tulokset ovat sen mukaisia.

Lähteet

A Brief History of CSS. 2016. Verkkoartikkeli. Viitattu 15.3.2020. https://www.css-class.com/a-brief-history-of-css/

A brief history of Node.js. N.d. Verkkoartikkeli. Viitattu 17.3.2020. https://nodejs.dev/a-brief-history-of-nodejs

An Introduction to JavaScript. 2020. Verkkoartikkeli. Viitattu 17.3.2020. https://javascript.info/intro

Baig, S. 2018. Vastaus keskustelufoorumilla. Viitattu 24.3.2020. https://www.quora.com/What-is-syntactic-sugar-in-programming-languages

Bemenderfer, J. 2017. Understanding Vue.js Lifecycle Hooks. Verkkoartikkeli. Viitattu 4.4.2020. https://alligator.io/vuejs/component-lifecycle/

Comparison with Other Frameworks. N.d. Vue:n verkkosivusto. Viitattu 19.4.2020. https://vuejs.org/v2/guide/comparison.html

Component State. N.d. Reactin viralliset verkkosivut. Viitattu 29.3.2020. https://reactjs.org/docs/faq-state.html

Copes, F. 2018a. Async vs sync code. Verkkoartikkeli. Viitattu 24.3.2020. https://flaviocopes.com/async-vs-sync/

Copes, F. 2018b. How to use Visual Studio Code. Verkkoartikkeli. Viitattu 30.3.2020. https://flaviocopes.com/vscode/

Copes, F. 2018c. The Vue Handbook. Verkkoartikkeli. Viitattu 4.4.2020. https://www.freecodecamp.org/news/the-vue-handbook-a-thorough-introduction-to-vue-js-1e86835d8446/

Creating Tables. N.d. Verkko-opas. Viitattu 29.3.2020. http://www.sql-course.com/create.html

Deleting Records. N.d. Verkko-opas. Viitattu 29.3.2020. http://www.sql-course.com/delete.html

Document Object Model. N.d. Verkko-opas. Viitattu 17.3.2020. https://www.javatpoint.com/document-object-model

Drop a Table. N.d. Verkko-opas. Viitattu 29.3.2020. http://www.sqlcourse.com/drop.html

ECMAScript 6. 2016. GitHub-repositorio. Viitattu 30.3.2020. https://github.com/lukehoban/es6features

Expo Introduction. N.d. Expon verkkosivut. Viitattu 1.4.2020. https://docs.expo.io/versions/latest/

Expo Limitations. N.d. Expon verkkosivut. Viitattu 5.4.2020. https://docs.expo.io/versions/latest/introduction/why-not-expo/

Express/Node introduction. 2020. Verkko-opas. Viitattu 18.3.2020. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

Freeman, J. 2019. What is JSON? A better format for data exchange. Verkkoartikkeli. Viitattu 19.3.2020. https://www.infoworld.com/article/3222851/what-is-json-a-better-format-for-data-exchange.html

Friesen, J. 2020. Android Studio for Beginners. Verkkoartikkeli. Viitattu 1.4.2020. https://www.javaworld.com/article/3095406/android-studio-for-beginners-part-1-installation-and-setup.html

Functional Components vs Class Components. N.d. Verkkoartikkeli. Viitattu 29.3.2020. https://guide.freecodecamp.org/react/functional-components-vs-class-components/

Gazarov, P. 2019. What is an API? In English, please. Verkkoartikkeli. Viitattu 19.3.2020. https://www.freecodecamp.org/news/what-is-an-api-in-english-please-b880a3214a82/

Gillis, A. 2018. Native App. Vastaus termimääritelmälle. Viitattu 25.4.2020. https://searchsoftwarequality.techtarget.com/definition/native-application-native-app

Google Firebase. N.d. Kuvakaappaus Firebase-kotisivuilta. Viitattu 14.4.2020. https://firebase.google.com/

Hamedani, M. 2018. What is Node.js? | Mosh. Lataaja Programming with Mosh. Video-opas YouTube-sivustolla. Viitattu 19.4.2020. https://www.youtube.com/watch?v=uVwtVBpw7RQ

Hoffman, J. 2017. A Look Back at the History of CSS. Verkkoartikkeli. Viitattu 15.3.2020. https://css-tricks.com/look-back-history-css/

HTML: Hypertext Markup Language. 2020. Verkko-opas. Viitattu 15.3.2020. https://developer.mozilla.org/en-US/docs/Web/HTML

Inserting into a Table. N.d. Verkko-opas. Viitattu 29.3.2020. http://www.sql-course.com/insert.html

Introduction to React. N.d. Verkko-opas. Viitattu 19.3.2020. https://survivejs.com/react/getting-started/introduction-to-react/

Introduction. 2020. Verkko-opas. Viitattu 17.3.2020. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction

JSON: What It Is, How It Works, & How to Use It. N.d. Verkkoartikkeli. Viitattu 19.3.2020. https://www.copterlabs.com/json-what-it-is-how-it-works-how-to-use-it/

Jöch, D. 2018. Functional vs Class-Components in React. Verkkoartikkeli. Viitattu 19.4.2020. https://medium.com/@Zwenza/functional-vs-class-components-in-react-231e3fbd7108

Lee, C. 2018. When (and why) you should use ES6 arrow functions — and when you shouldn't. Verkkoartikkeli. Viitattu 19.4.2020. https://www.freecodecamp.org/news/when-and-why-you-should-use-es6-arrow-functions-and-when-you-shouldnt-3d851d7f0b26/

Lundberg, A. 2019. Color meanings and the art of using color symbolism. Verkkoartikkeli. Viitattu 12.4.2020. https://99designs.com/blog/tips/color-meanings/

Maggini, G. 2019. Deciding between native and cross-platform mobile frontend programming frameworks. Verkkoartikkeli. Viitattu 25.4.2020. https://developer.ibm.com/technologies/mobile/articles/deciding-between-native-and-cross-platform-mobile-frontend-programming-frameworks/

McGinnis, T. 2019. Why React Hooks? Verkko-opas. Viitattu 25.4.2020. https://tylermcginnis.com/why-react-hooks/

NoSQL Databases Explained. N.d. Verkkoartikkeli. Viitattu 29.3.2020. https://riak.com/resources/nosql-databases/index.html?p=9937.html

Nowak, M. 2019. Vue vs React in 2020: Which Framework to Choose and When. Verkkoartikkeli. Viitattu 19.4.2020. https://www.monterail.com/blog/vue-vs-react-2020

Petty, J. N.d. What is Unity 3D & What is it Used For? Verkkoartikkeli. Viitattu 19.4.2020. https://conceptartempire.com/what-is-unity/

Python Object Oriented Programming. N.d. Verkkoartikkeli. Viitattu 17.3.2020. https://www.programiz.com/python-programming/object-oriented-programming

ReactJS Tutorial - 8 - JSX. 2018. Video-ohje YouTube-sivustolla. Lataaja Codevolution. Viitattu 24.3.2020. https://www.youtube.com/watch?v=7fPXI_MnBOY

Sandoval, K. 2018. What is the Difference Between an API and an SDK? Verkkoartikkeli. Viitattu 12.4.2020. https://nordicapis.com/what-is-the-difference-between-an-api-and-an-sdk/

Selecting Data. N.d. Verkko-opas. Viitattu 29.3.2020. http://www.sqlcourse.com/select.html

Sign your app. 2019. Android Studio -dokumentaatio. Viitattu 26.4.2020 https://developer.android.com/studio/publish/app-signing

State and Lifecycle. N.d. Verkko-opas. Viitattu 30.3.2020. https://reactjs.org/docs/state-and-lifecycle.html

Stevens, E. 2019. What Is User Experience (UX) Design? Everything You Need To Know To Get Started. Verkkoartikkeli. Viitattu. 12.4.2020. https://careerfoundry.com/en/blog/ux-design/what-is-user-experience-ux-design-everything-you-need-to-know-to-get-started/

Stevenson, D. 2018. What is Firebase? The complete story, abridged. Verkkoartikkeli. Viitattu 12.4.2020. https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0

Styling and CSS. N.d. Reactin verkkosivusto. Viitattu 19.4.2020. https://reactjs.org/docs/faq-styling.html

Tasanen, P. 2019. Mitä integraatio, rajapinta ja api tarkoittavat? Verkkoartikkeli. Viitattu 18.3.2020. https://www.valjas.fi/mita-integraatio-rajapinta-ja-api-tarkoittavat/

Three Principles. N.d. Reduxin verkkosivut. Viitattu 5.4.2020. https://redux.js.org/introduction/three-principles#single-source-of-truth

Traversy, B. 2019a. VueJS Crash Course. Video-opas YouTube-sivustolla. Viitattu 4.4.2020. https://www.youtube.com/watch?v=Wy9q22isx3U

Traversy, B. 2019b. Vuex Crash Course. Video-opas YouTube-sivustolla. Viitattu 5.4.2020. https://www.youtube.com/watch?v=5lVQgZzLMHc

Traversy, B. 2020. React Native Crash Course. Video-opas YouTube-sivustolla. Viitattu 5.4.2020. https://www.youtube.com/watch?v=Hf4MJH0jDb4

Työelämän tutkiva kehittämistoiminta. N.d. Opinnäytetyön ohjaajan käsikirja. Viitattu 15.3.2020. https://oppimateriaalit.jamk.fi/yamk-kasikirja/tyoelaman-tutkiva-kehittamistoiminta/

Updating Records. N.d. Verkko-opas. Viitattu 29.3.2020. http://www.sql-course.com/update.html

User Experience Basics. N.d. Verkkoartikkeli. Viitattu 12.4.2020. https://www.usability.gov/what-and-why/user-experience.html

User Interface Design in Modern Web Applications. 2011. Verkkoartikkeli. Viitattu 12.4.2020. https://www.smashingmagazine.com/user-interface-design-in-modern-web-applications/

Web Components. 2019. Verkko-opas. Viitattu 19.3.2020. https://developer.mozilla.org/en-US/docs/Web/Web_Components

Wells, R. 2019. What Is an Emulator? Verkkoartikkeli. Viitattu 1.4.2020. https://www.lifewire.com/what-is-an-emulator-4687005

What is Database? What is SQL? N.d. Verkkoartikkeli. Viitattu 24.3.2020. https://www.guru99.com/introduction-to-database-sql.html

What is MongoDB? N.d. Verkkoartikkeli. Viitattu 5.4.2020. https://www.guru99.com/what-is-mongodb.html

What is npm? 2011. Verkkoartikkeli. Viitattu 14.4.2020. https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/

What is open source? N.d. Verkkoartikkeli. Viitattu 14.4.2020. https://www.redhat.com/en/topics/open-source/what-is-open-source

What is SQL? N.d. Verkko-opas. Viitattu 29.3.2020. http://www.sqlcourse.com/intro.html

What is Vuex? N.d. Vuex:n verkkosivusto. Viitattu 5.4.2020. https://vuex.vuejs.org/vuex.png

What's the Difference Between the Front-End and Back-End? 2015. Verkkoartikkeli. Viitattu 14.4.2020. https://www.pluralsight.com/blog/film-games/whats-difference-front-end-back-end

World Wide Web. N.d. Kuvakaappaus ensimmäisestä verkkosivusta. Viitattu 15.3.2020. http://info.cern.ch/hypertext/WWW/TheProject.html